AUTHOR: **Antony Holmes**

DEGREE: .....

TITLE: **Understanding morphogenesis in myxobacteria from a theoretical and experimental perspective**

DATE OF DEPOSIT: .....

I **agree** that this thesis shall be available in accordance with the regulations governing the University of Warwick theses.

I **agree** that the summary of this thesis may be submitted for publication.

I **agree** that the thesis may be photocopied (single copies for study purposes only).

Theses with no restriction on photocopying will also be made available to the British Library for microfilming. The British Library may supply copies to individuals or libraries, subject to a statement from them that the copy is supplied for non-publishing purposes. All copies supplied by the British Library will carry the following statement:

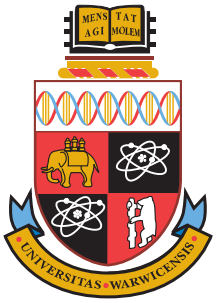
> "Attention is drawn to the fact that the copyright of this thesis rests with its author. This copy of the thesis has been supplied on the condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the author's written consent."

AUTHOR'S SIGNATURE: .....

USER DECLARATION

1. I undertake not to quote or make use of any information from this thesis without making acknowledgement to the author.
2. I further undertake to allow no-one else to use this thesis while it is in my care.

DATE	SIGNATURE	ADDRESS
.....	.....	.....
.....	.....	.....
.....	.....	.....
.....	.....	.....
.....	.....	.....



Understanding morphogenesis in myxobacteria from a theoretical and experimental perspective

by

Antony Holmes

Thesis
Submitted to the University of Warwick
for the degree of
Doctor of Philosophy

MOAC Doctoral Training Centre
September, 2009

THE UNIVERSITY OF WARWICK

Contents

List of Figures vi
List of Tables x
Acknowledgements xii
Declaration xiii
Abbreviations xv
Conventions used in this thesis xvi
Abstract xvii
1. Introduction 1
1.1. Motivations and related work 1
1.1.1. Developing realistic models 2
1.1.2. Developing a comprehensive model of myxobacterial behaviour 3
1.1.3. Developing modelling tools 3
1.2. Contribution 3
1.3. Thesis outline 4
2. The biology of myxobacteria 8
2.1. Introduction 8
2.2. Biology 10
2.3. Distribution in nature 11
2.4. Inter-cellular signalling 12
2.5. Motility 13
2.6. Pili and fibrils 14
2.7. The life-cycle of myxobacteria 15
2.7.1. Vegetative cells 15
2.7.2. Rippling 16
2.7.3. Streaming 17
2.7.4. Fruiting body formation 18
2.7.5. Sporulation 22

2.8. Population level cheating 24
2.9. Novel uses of myxobacteria 25
2.9.1. Cancer treatment 25
2.9.2. Stone restoration 25
3. Computation models of biological systems **27**
3.1. Introduction 27
3.2. The modelling process 29
3.3. Modelling in a biological context 30
3.4. Cellular automata 32
3.4.1. Neighbourhoods 35
3.4.2. Lattice gas cellular automata 36
3.4.3. State transition 37
3.5. Monte Carlo methods 38
3.5.1. Equilibrium 40
3.5.2. Fluctuation 41
3.5.3. Isling model 41
3.5.4. Monte Carlo method 42
3.5.5. The Metropolis algorithm 45
3.6. The Potts model 46
3.7. Cellular Potts model 48
3.8. Discussion 48
4. The role of phosphate during development **50**
4.1. Introduction 50
4.2. Materials and methods 51
4.2.1. Growth media 52
4.2.2. Phosphate growth media 52
4.2.3. Stock solutions 55
4.2.4. *M. xanthus* DK1622 resurrection 55
4.2.5. Culture preparation 56
4.2.6. Measurement of cell growth 56
4.3. Results 57
4.4. Discussion 61
5. The design of FABCell **66**
5.1. Introduction 67
5.2. Requirements analysis 68
5.3. Existing simulation tools 70
5.4. System design 72
5.4.1. Architecture 73
5.4.1.1. Core FABCell components 73

5.4.2. The interaction step . . . 79
5.4.3. Precondition handling . . . 80
5.4.4. Neighbourhoods . . . 81
5.4.5. Creating a model . . . 85
5.5. Design patterns . . . 85
5.6. Parallelism . . . 89
5.6.1. Implementing concurrency in C++ . . . 89
5.6.2. A formal expression of concurrency . . . 95
5.7. Creating a user interface . . . 96
5.7.1. Model specification interface . . . 98
5.8. Plug-in Framework . . . 99
5.8.1. Object caches . . . 102
5.9. Visualisation . . . 103
5.10. Testing . . . 106
5.11. Case studies . . . 108
5.11.1. Game of Life . . . 108
5.11.2. A cellular Potts model approach to morphogenesis . . . 110
5.12. Discussion . . . 110
6. A developmental model of FrzS translocation 114
6.1. Introduction . . . 114
6.2. Background . . . 115
6.3. An oscillatory model of FrzS migration . . . 118
6.4. The *Motilator* model of FrzS migration . . . 122
6.5. Model implementation . . . 123
6.6. Stability analysis . . . 124
6.7. Results . . . 130
6.7.1. Signalling and reversing . . . 130
6.7.2. Oscillations . . . 130
6.7.3. Agent based simulation . . . 134
6.8. Discussion . . . 139
7. A Monte Carlo approach to modelling cell dynamics 143
7.1. Introduction . . . 143
7.2. Background . . . 144
7.3. A revised model of the Frz transduction pathway . . . 146
7.4. An off-lattice approach . . . 149
7.4.1. A multi-component model approach . . . 156
7.5. Simulating cell physics . . . 156
7.5.1. Alignment energy . . . 158
7.5.2. Bending energy . . . 158
7.5.3. Climbing energy . . . 159

7.5.4. Collision energy . . . 160
7.5.5. Gravitational energy . . . 161
7.5.6. Propulsion energy . . . 161
7.5.7. Slime trail following energy . . . 162
7.5.8. Stretching energy . . . 163
7.5.9. C-signalling . . . 164
7.5.10. Simulation parameters . . . 164
7.6. Results . . . 164
7.6.1. Motility . . . 165
7.6.2. Slime trails . . . 165
7.6.3. The effects of slime trail following . . . 166
7.6.4. The effects of climbing . . . 168
7.6.5. Collision avoidance . . . 170
7.6.6. Signalling and reversing . . . 171
7.6.7. Ripple formation . . . 171
7.6.8. From rippling to streaming . . . 175
7.7. Discussion . . . 180
8. Fruiting body morphogenesis **184**
8.1. Introduction . . . 184
8.2. Modelling fruiting body formations . . . 188
8.2.1. Myxobacterial cell design . . . 188
8.2.2. Simulation volume . . . 189
8.2.3. Cell influx . . . 190
8.3. Algorithm implementation . . . 191
8.4. System Hamiltonian . . . 191
8.4.1. Adhesion energy . . . 194
8.4.2. Climbing energy . . . 195
8.4.3. Gravitational energy . . . 197
8.5. A model of sporulation . . . 198
8.6. Results . . . 199
8.6.1. Model of fruiting body formation . . . 199
8.6.2. Fruiting with a finite number of cells . . . 199
8.6.3. Climbing . . . 200
8.6.4. Cell adhesion . . . 202
8.6.5. Aggregate formation . . . 202
8.6.6. Fruit dispersal . . . 204
8.6.7. Stable fruiting body formation . . . 208
8.6.8. Sporulation . . . 209
8.7. Discussion . . . 214

9. Conclusion **218**
9.1. Phosphate usage . . . 219
9.2. Developing tools to study myxobacteria morphogenesis. . . . 220
9.3. A new understanding of the Frz pathway . . . 221
9.4. Modelling cell physics . . . 222
9.4.1. Model realism . . . 222
9.5. Fruiting . . . 223
9.6. A comprehensive approach. . . . 224
9.7. Future Work . . . 225
9.7.1. A biological perspective . . . 225
9.7.2. A computational perspective . . . 226
A. Supplementary movies **228**
B. *Motilator* equations **231**
B.1. General form of the Jacobian matrix for the *Motilator* . . . 231
B.2. Jacobian matrix for the *Motilator* . . . 231
C. MATLAB® implementation of the *Frzilator* **233**
C.1. ODE system . . . 233
C.2. ODE step . . . 233
C.3. System response . . . 234
D. MATLAB® implementation of the *Motilator* **239**
D.1. ODE system . . . 239
D.2. ODE step . . . 240
E. Wolfram rule 110 cellular automaton **243**
E.1. XML model definition . . . 243
F. Instructions for compiling FABCell **251**
F.1. Obtaining the source code . . . 251
F.2. Compiling . . . 251
F.3. Running a model. . . . 252
F.4. Visualising a model . . . 253
F.5. Example models . . . 256
Bibliography **258**
Index **278**
Colophon **281**

List of Figures

1.1. Thesis chapter map 5

2.1. Electron micrographs of extracellular appendages 14
2.2. The life-cycle of myxobacteria 15
2.3. Rippling cells 17
2.4. Streaming cells 18
2.5. Fruiting body formation 19
2.6. Examples of fruiting body formations 23

3.1. Lattice connectivity 34
3.2. Generalised Potts model 47

4.1. Protocol for *M. xanthus* resurrection from freezer stocks 56
4.2. Protocol for taking optical density measurements 57
4.3. The effects of phosphate sources on the growth of *M. xanthus* . . . 58
4.4. Phosphate usage experiment set one 58
4.5. Phosphate usage experiment set two 59
4.6. Phosphate usage experiment set three 60
4.7. Phosphate usage experiment set four 61
4.8. Phosphate sources identified as causing growth in *M. xanthus* . . . 62

5.1. FABCell use case requirements 69
5.2. FABCell program library structure 73
5.3. Class diagram of the core components of FABCell 74
5.4. Class diagram of the peripheral components of FABCell 75
5.5. Cell representation within FABCell 77
5.6. Space representation within FABCell 78
5.7. The FABCell program loop 82
5.8. Neighbourhood classifications. 83
5.9. Relative neighbourhood algorithm 85

5.10. Flowchart of the steps required to create a model 86
5.11. Fork model of parallelism used by FABCell. 92
5.12. FABCell OpenMP parallel enabled code 94
5.13. The FABCell user interface hierarchy 97
5.14. Excerpt of XML model specification 99
5.15. C++ code snippet to create a cellular automaton 100
5.16. Loading a plug-in using XML 101
5.17. Using the FABCell object cache. 102
5.18. FABCell viewer graphics pipeline 104
5.19. The software testing process 107
5.20. Simulation of The Game of Life 109
5.21. A simulation of cell sorting using a Cellular Potts Model 111

6.1. Schema of the Frz signalling and the *Motilator* 119
6.2. Theoretical protein transport model 121
6.3. The effect of varying s_1 for solution p_1 128
6.4. The effect of varying x_1 for solution p_1 128
6.5. The effect of varying s_1 for solution p_5 129
6.6. The effect of varying x_1 for solution p_5 129
6.7. Refractory period characteristics of the *Motilator* 131
6.8. The response of the *Motilator* to different levels of MglA* signalling. 132
6.9. Response of the *Motilator* showing FrzS movement during a ripple cycle 133
6.10. LGCA model for simulating the *Motilator* in a cell population 134
6.11. Variance of the proximity function 136
6.12. Space time plot showing the evolution of an agent based model controlled by the *Motilator*. 137
6.13. Density plots of cell movement in the xy-plane for a *Motilator* simulation of 45,000 cells 138

7.1. Schema of the *Phys-Motilator* 148
7.2. Principle physical characteristics of *M. xanthus* cells 151
7.3. Myxobacteria cell flexibility 152
7.4. Schema of a segmented model cell 154
7.5. Simulation volume constraints 155
7.6. A simulation consists of an object stack representing each entity and feature present in the environment 156
7.7. Physical characteristics of *Phys-Motilator* cells 166
7.8. Comparison of the ability of the *Phys-Motilator* to form ripples when slime trail following is disabled and enabled 167

7.9. The effects of slime trail following on local cell alignment in a ripple 168
7.10. Space time plot showing the effects of cell climbing on ripple formation after 280 min 169
7.11. The effects of cell climbing on cell proximity during rippling 170
7.12. Ripple formation using an off lattice simulation of 5500 cells with an unlimited nutrient source 172
7.13. Space time plots of ripple formation using the *Phys-Motilator* with an unlimited nutrient source 173
7.14. Space time plots showing the transition from *pre-ripplingto* rippling 173
7.15. Histograms of cell reversal times during rippling 175
7.16. The effect of FrzG on the *Phys-Motilator* 176
7.17. The effect of varying MglA on the *Phys-Motilator* 177
7.18. Histograms of cell reversal frequencies during the transition from rippling to streaming 178
7.19. Space time plots of 5500 simulated cells showing the transition from rippling to streaming 179

8.1. The main stages of fruiting body formation 185
8.2. Cell design for fruiting body simulations 189
8.3. Converging stream formations 190
8.4. A model of cell climbing and layer formation 196
8.5. Fruiting body formation with a finite number of cells. 201
8.6. The effects of adhesion on stream formation 203
8.7. Simulation of 300 cells using the fruiting Hamiltonian and variant of the *Motilator* which is non reversing 204
8.8. Top-down view of fruiting body development. 205
8.9. Space time plots showing the effect of low cell influx on fruiting body formation. 206
8.10. Three-dimensional plots showing the how fruiting body formations can spontaneously form and reform 207
8.11. Stable fruiting body development in the xy-plane. 208
8.12. Space time plots showing fruiting body formation 210
8.13. Average cell counts of simulated motile and spore cells 211
8.14. Three-dimensional view of fruiting body simulation when cells are allowed to sporulate 212
8.15. View of a fruiting body simulation after 24 h where cells were allowed to sporulate. 213
8.16. The localisation of myxospores and C-signal within a fruiting body simulation 214

C.1. Oscillations of the *Frzilator* 235
C.2. *Frzilator* response to increases in C-signalling duration 236
C.3. Stable *Frzilator* response to sliding 20 min signalling window during the streaming phase 237
C.4. Unstable *Frzilator* response to sliding 20 min signalling window during the streaming phase 238

List of Tables

1.1. Existing models of the myxobacterial life-cycle 2
1.2. Models created for the thesis to study the myxobacteria life-cycle. 4

3.1. Uses of models 28
3.2. Modelling terminology 28
3.3. Approaches to modelling spatio-temporal pattern formation 31

4.1. DCY and TPM growth media for initial cell cultures 53
4.2. A1W and A1W-P minimal growth media 53
4.3. M1 and M1-P growth media 54
4.4. Phosphate sources tested in A1W-P medium 55

5.1. Existing biological simulation tools. 71
5.2. FABCell viewer object types 103
5.3. FABCell viewer display modules 105

6.1. Parameter values for the *Motilator* 124
6.2. Solution sets for the intersections of the nullclines 126

7.1. Parameter values for the *Phys-Motilator* models 150
7.2. The role of specific energy terms in the myxobacteria rippling Hamiltonian 157
7.3. Parameters for the off-lattice simulation of *M. xanthus* 165

8.1. Parameters for models used to simulate fruiting body formation in *Myxococcus xanthus* 193
8.2. The role of specific energy terms in the fruiting body Hamiltonian . 193

A.1. Myxobacteria movies 228
A.2. FABCell movies 229

F.1. FABCell dependencies 252

F.2. Files for FABCell model instantiation 253
F.3. Keyboard controls for the FABCell viewer 254
F.4. Command line options for the FABCell viewer 255
F.5. FABCell models 256

Acknowledgements

First, I would like to express my deepest gratitude to my supervisors Dr Sara Kalvala and Dr David Whitworth, for their generous support and guidance throughout the research. I would also like to thank Prof. Alison Roger and the administrators of MOAC for their support and encouragement and the EPSRC for funding throughout my PhD.

Many thanks go to Lahari de Alwis for proof reading the thesis and making helpful suggestions. I would also like to mention James Lister Sinfield for his friendship and interesting discussions.

Special thanks are due to Yi for his great friendship for the past five years and his support and encouragement. I would also like to express my deepest gratitude to Ioana for her wonderful friendship and caring nature. Her dedicated work ethic inspired me to persevere. I could not have wished for two better friends and I am indebted to them for the warmth and love they brought into my life.

Last but not least, I would like to thank my family for their support and being there for me.

Declaration

This thesis is presented in accordance with the regulations for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree. The work in this thesis has been undertaken by myself except where otherwise stated. All quoted text remains the copyright of the original attributed author and all trademarks are acknowledged.

Parts of the work presented in this thesis have been used for publication. The phosphate assay conducted in Chapter 4 was published as "Phosphate acquisition components of the Myxococcus xanthus pho regulon are regulated by both phosphate availability and development" in the *Journal of Bacteriology* [191]. The *Motilator* model presented in Chapter 6 forms part of a research article entitled "A model of dynamic protein relocalisation explains several features of myxobacterial motility and development", which has been submitted for publication. The statistical physics model of cell motility presented in Chapter 7 was presented at *the 2nd International Conference on Bioinformatics and Systems Biology (BSB 2009)* and published as "Myxobacteria motility: a novel 3D model of rippling behaviour in *Myxococcus xanthus*" in *Communications of the Systematics and Informatics World Network* [60]. The fruiting model presented in Chapter 8 forms a journal article entitled "Spatial simulations of myxobacterial development", which has been submitted for publication.

The FABCell software is available on the `CD-ROM` accompanying this thesis or from the FABCell website: `http://sourceforge.net/projects/fabcell/`. Models that generated published data are not available from the website but are included on the `CD-ROM` for reference (see Appendix F).

Every care was taken when referencing websites to ensure only major websites concerned with a particular subject were included to minimise the possibility of them subsequently becoming invalid. All web addresses were valid as of September, 2009.

The index has not been constructed to a professional standard and should not be relied on for completeness. It is hoped, however, that the index may help readers find information of interest more easily.

Antony Holmes

September, 2009

Abbreviations

API	Application Programming Interface
CA	Cellular Automata
CFU	Colony-forming unit
CPM	Cellular Potts Model
CPU	Central Processing Unit
DCY	Double Casitone Yeast growth media
DNA	Deoxyribonucleic acid
E. coli	*Escherichia coli*
EPS	Extra-cellular Polysaccharide Slime matrix
FABCell	Framework for Agent Based Cell modelling
GFP	Green Fluorescent Protein
I/O	Input/Output
MPI	Message Passing Interface
ODE	Ordinary Differential Equation
OO	Object-oriented
PDE	Partial Differential Equation
TPM	Tris Phosphate Magnesium growth media
M. xanthus	*Myxococcus xanthus*
UML	Unified Modelling Language
XML	Extensible Markup Language

Conventions used in this thesis

`Constant width` is used for:

- Anything that might appear in an XML document including element names, tags, attribute values and processing instructions.
- Anything that might appear in a program, including code fragments, keywords, operators, method names, class names and literals.
- File names and directories on the `CD-ROM` accompanying the thesis.

Italic is used for:

- New terms where they are defined.
- Emphasis in body text particularly of terms with a specific meaning in the context they are being used (e. g. *rippling*).

Experimental results and data are displayed plainly within figures whilst boxed and titled figures convey information and concepts. Significant code fragments, complete programs and documents are generally placed in their own figure.

Abstract

Several species of bacteria exhibit multicellular behaviour, with individuals cells co-operatively working together within a colony. Often this has communal benefit since multiple cells acting in unison can accomplish far more than an individual cell can and the rewards can be shared by many cells.

Myxobacteria are one of the most complex of the multicellular bacteria, exhibiting a number of different spatial phenotypes. Colonies engage in multiple emergent behaviours in response to starvation culminating in the formation of massive, multicellular fruiting bodies.

In this thesis, experimental work and theoretical modelling are used to investigate emergent behaviour in myxobacteria. Computational models were created using FABCell, an open source software modelling tool developed as part of the research to facilitate modelling large biological systems.

The research described here provides novel insights into emergent behaviour and suggests potential mechanisms for allowing myxobacterial cells to go from a vegetative state into a fruiting body. A differential equation model of the Frz signalling pathway, a key component in the regulation of cell motility, is developed. This is combined with a three-dimensional model describing the physical characteristics of cells using Monte Carlo methods, which allows thousands of cells to be simulated. The unified model explains how cells can ripple, stream, aggregate and form fruiting bodies. Importantly, the model copes with the transition between stages showing it is possible for the important myxobacteria control systems to adapt and display multiple behaviours.

Chapter 1

Introduction

Myxobacteria are one of the most complex and fascinating members of the bacterial domain. Despite over 100 years of active research, a well-characterised understanding of the complex signalling pathways and cellular systems that control cell behaviour remains elusive.

Computational and mathematical modelling approaches to cellular behaviour have started to reveal how myxobacteria might be communicating and coordinating their behaviour. Importantly, these techniques reveal that although many of the systems are biologically complex, their functions are relatively simple and produced via multiple simple systems working cooperatively.

1.1 Motivations and related work

The myxobacterial life-cycle is often described as a series of discrete phases which cells enter, but it is a continuous process with a transition from one phase to another. Myxobacteria have four major life-cycle events (discussed more extensively in Chapter 2): rippling, streaming, fruiting and sporulation. Existing models of behaviour typically consider one phase and produce a model to capture a very specific behaviour. The models themselves can be categorised as being either lattice based (cellular automata), off-lattice or mathematical (primarily ODE and PDE based). Table 1.1 summarises the major existing models by the life-cycle phase they model and their type. Few of the models look at morphogenesis in myxobacteria and these are predominantly lattice

based, which focus on rippling. The models developed in this thesis bridge some of the gaps particularly with regard to providing more mathematical and off-lattice models of cellular behaviour, since ultimately models should be as detailed as possible. An important secondary goal was to create more unified models that explain the life-cycle more comprehensively.

Table 1.1: Existing models of the myxobacteria life-cycle. Models are classified by their type. Dashes indicate no suitable published models exist.

Type	Lattice	Off-lattice	Mathematical
Rippling	Alber et al. [2], Börner et al. [21]	—	Igoshin et al. [63], Igoshin and Oster [61]
Streaming	Sozinova et al. [160]	Starruß et al. [165], Wu et al. [200, 202]	—
Fruiting	Hendrata and Birnir [57], Sozinova et al. [161]	—	—
Sporulation	Sozinova et al. [161]	—	—

1.1.1 Developing realistic models

Existing theoretical models of myxobacterial phenomena have a number of limiting factors. Cellular Automata (CA) models have been used to simulate large numbers of cells but at the expense of realism; cells have rigid geometries and restricted paths of motion providing a very coarse-grained approximation of movement. The work presented here was motivated by a desire to model cells more realistically and capture more of their biological properties.

The physical number of cells simulated is also an issue with some existing models. Hendrata and Birnir [57], for example, present a model of fruiting body formation using approximately 300 cells. A real fruiting body has upwards of 100,000 cells so there are questions over how well this model approximates reality. The number of cells plays an important role in determining behaviour; cells can use a form of quorum sensing to decide when to undertake particular actions. A few hundred cells may not be enough to trigger population level events, so that any model using a relatively small number of cells might require artificial inducement to exhibit the desired behaviour, somewhat negating the useful conclusions that can be drawn from it. Though

computational resources will always impose bounds on the complexity of any model, it would be desirable to create simulations on a much larger scale.

1.1.2 Developing a comprehensive model of myxobacterial behaviour

Existing models of behaviour tend to focus on specific life-cycle events; however, myxobacteria exhibit multiple behaviours (often in succession) and this is not considered fully. Certain models of rippling (see Chapter 2 for more details) show how cells can coordinate into ripples; however, cells ripple indefinitely and the models do not explain how cells alter their behaviour to transition into subsequent stages of the life-cycle. Ideally, a sufficiently detailed model would be able to cope with all of the observed behaviours [2, 21].

1.1.3 Developing modelling tools

A literature review highlighted that a number of existing models appear to obfuscate details of exactly how they were implemented, making it difficult to reproduce the results or use the model as a basis for new models. The specific simulation tools used are rarely mentioned and implementation details are often omitted. There is a gulf of execution between how a model is described in abstract terms mathematically and how it is actually implemented in software. Part of the work presented here was designed to allow the author to understand how modelling tools actually work and create a consistent, well defined framework on which to create models.

1.2 Contribution

The thesis contributes to the understanding of myxobacterial behaviour from an experimental and theoretical perspective.

The main theoretical contribution of this thesis is a number of models (listed in Table 1.2) to explain emergent behaviour in myxobacteria and also how these models can be linked to form a comprehensive model of the life-cycle. The models address aspects of morphogenesis that are not covered in the existing, published models given in Table 1.1 and provide a more detailed analysis of the life-cycle stages.

Table 1.2: Models created for the thesis to study the myxobacteria life-cycle.

Type	Lattice	Off-lattice	Mathematical
Rippling	*Motilator* (Chapter 6)	*Phys-Motilator* (Chapter 7)	*Motilator* (Chapter 6)
Streaming	—	*Phys-Motilator* (Chapter 7)	*Phys-Motilator* (Chapter 7)
Fruiting	—	*Fruit-Motilator* (Chapter 8)	—
Sporulation	—	*Fruit-Motilator* (Chapter 8)	—

1.3 Thesis outline

The thesis is comprised of nine chapters: an introduction, two preliminary chapters, five results chapters and a conclusion. The chapters are connected and should not be considered in isolation (see Figure 1.1). The models developed in Chapters 6, 7 and 8 are dependent on the FABCell simulation tool developed in Chapter 5.

Chapter 2 provides an overview of relevant background material on the biology of myxobacteria. Chapter 3 provides an overview of computational methods for simulating biological systems with a focus on Monte Carlo methods and mathematical modelling techniques, which are used in Chapters 6 to 8. Chapter 4 discusses phosphate usage in *Myxococcus xanthus*. Chapter 5 discusses the implementation of a modelling framework to simulate the models described in Chapters 6 to 8. Chapter 6 discusses a model of protein translocation and how it can be used to explain emergent behaviour. Chapter 7 extends the work of Chapter 6 to discuss the role of nutrition in the myxobacterial life-cycle and how Monte Carlo dynamics can be used to create a physical model of cell dynamics which can be coupled to the protein translocation model. Chapter 8 refines the Monte Carlo dynamics further to look at fruiting body formation. Together, Chapters 6 to 8 show that a Monte Carlo model can be used to explain all stages of the myxobacterial life-cycle and how cells can display multiple spatial pattern formations using a set of signalling pathways.

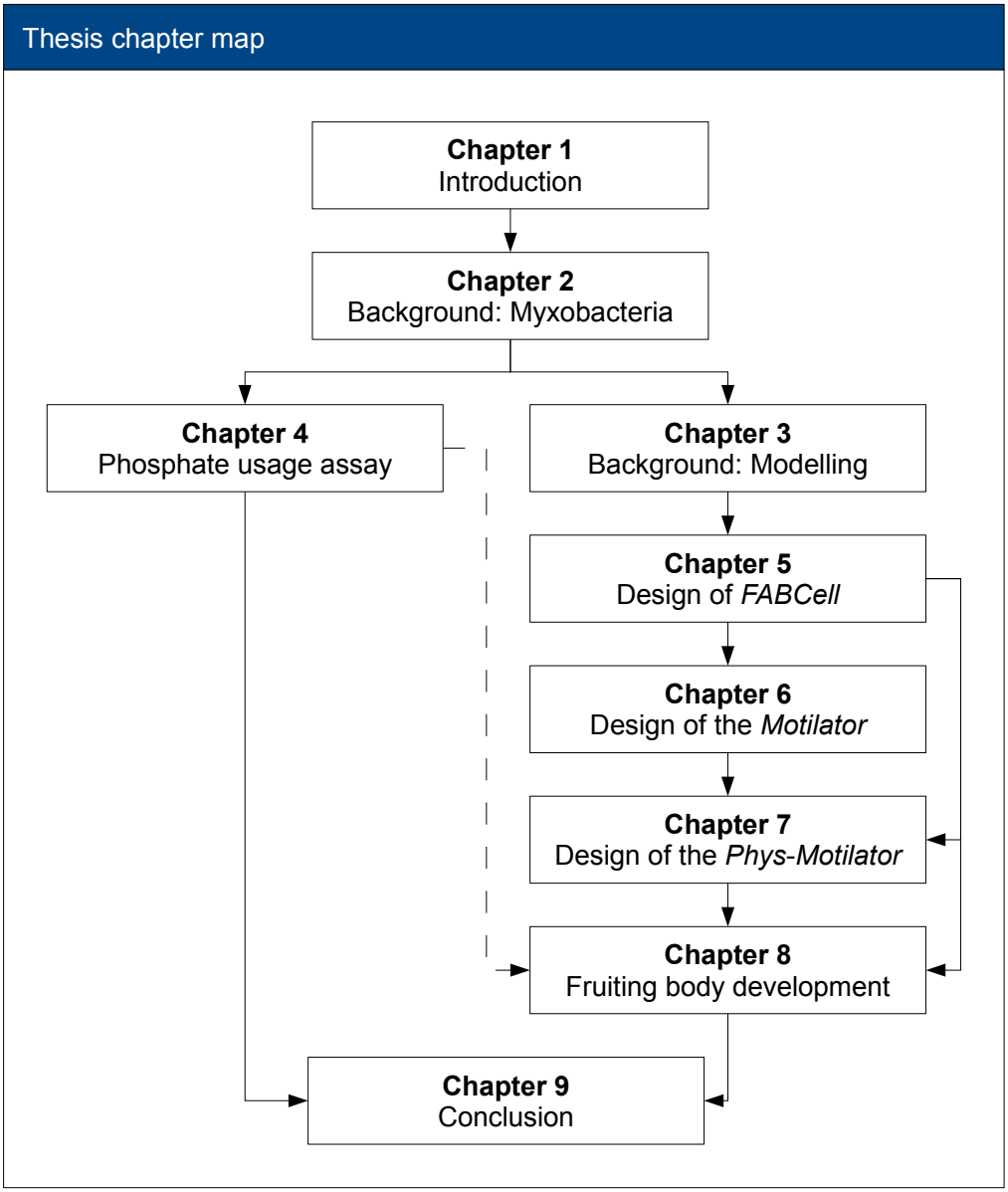


Figure 1.1: Thesis chapter map. A visual summary of the chapters of the thesis and how they relate to each other. Solid lines with arrowhead imply dependence, dashed lines indicate potential dependence. The phosphate assay forms a separate and parallel stream of research to the modelling work.

Chapter 2. Provides background knowledge of myxobacteria and their behaviour for the proceeding results chapters. It is intended to supplement introductory information presented in the results chapters, which, for brevity, only provide background material pertinent to the specific aspect of the life-cycle being discussed.

Chapter 3. Provides background knowledge on computational modelling techniques for simulating biological systems. The theory of Monte Carlo methods is explained in detail, showing how they can be adapted from a purely physical perspective to look at biological problems.

Chapter 4. A discussion of the role of phosphate as a nutrient source for myxobacteria. Chapter 7 discusses the role of nutrition in the emergent phenomenon of rippling.

Chapter 5. A discussion of the design and implementation of the Framework for Agent Based Cellular modelling (FABCell), a software modelling tool kit developed in C++. It provides a concrete implementation of the model theory so that experiments based on the models could be performed. No existing software fitted the needs of this research so FABCell was created. This part of the project required a proper design rationale and structured management to cope with FABCell's large code base. The software is robust enough to be used as a standalone modelling tool for other projects. It has been released as an open source tool for the research community.

Chapter 6. Myxobacteria display multiple spatial phenotypes which are thought to be attributable to the Frz signalling pathway. This chapters discusses the construction of a mathematical model called the *Motilator*, which explains how the translocation of FrzS between cell poles can be used as a triggering mechanism to instigate rippling behaviour.

Chapter 7. An extension and complement to Chapter 6 which discusses how a model describing the physics of myxobacteria can be combined with the *Motilator* to explain rippling. The chapter extends the *Motilator* to cope with the effects of nutrition and

shows how cells can go from a pre-rippling vegetative state into a rippling state and finally into a streaming state.

Chapter 8. A further refinement to the rippling model in Chapter 7 shows how cells can be made to aggregate and form fruiting bodies. Combined with the two previous chapters, a comprehensive model of myxobacteria is developed showing how all the major stages of the myxobacterial life-cycle can be explained using a single, unified model.

Chapter 9. A summary of the research presented in the thesis which aims to bring together the results chapters. The thesis finishes with a discussion of potential directions for future research as well as how the FABCell software can be improved.

Chapter 2

The biology of myxobacteria

This chapter is a brief introduction to the biology of myxobacteria and gives an overview of the life-cycle. The numerous social and spatial phenotypes cells exhibit provide the motivation for subsequent chapters that model certain key phenomena. The chapter contains a more detailed review of cell biology than in other chapters, which instead focus on specific aspects of the life-cycle.

2.1 Introduction

Bacteria remain one of the most abundant organisms on Earth ubiquitous in all environments with approximately forty million bacterial cells in a gram of soil and a million cells in a millilitre of fresh water. They constitute a large proportion of the world's biomass and play an important role in all ecological systems. Their role in the life-cycle of all organisms cannot be emphasised enough and understanding the functions of bacteria and how they interact with other organisms can provide crucial understanding in how organisms function and the power bacteria hold over them.

Bacteria are single cell organisms lacking a nucleus but capable of maintaining pockets of DNA or plasmids which can be replicated along with the cell's own DNA. Bacteria are divided into two major functional groups: Gram-negative and Gram-positive. This is based on the Gram staining test which reveals structural information about a cell. Cells categorised as Gram-negative typically do not retain a crystal violet dye used in staining.

Myxobacteria are Gram-negative members of the δ-proteobacteria. Gram-positive bacteria will retain the crystal violet dye when washed in a decolourising solution. In a Gram stain test, a counter stain (commonly safranin) is added after the crystal violet, colouring all Gram-negative bacteria a red or pink colour. The test itself is useful in classifying two distinct types of bacteria based on structural differences in their cell walls. Gram negative cells are characterised by the following properties:

1. A cytoplasmic membrane.
2. A space between the layers of peptidoglycan and the secondary cell membrane called the periplasm.
3. Cells do not sporulate in general.

The myxobacteria are a family of bacteria that are perhaps one of the most fascinating for their somewhat idiosyncratic behaviour that is atypical of other, simpler bacteria. Cells are characterised by a complex developmental life-cycle and social behaviour normally associated with higher organisms like eukaryotes. It is these behaviour patterns that have captivated the attention of microbiologists for the last century as even with the extensive research that has been carried out, we are only just beginning to piece together exactly what each of the genes do and how all of the signalling pathways and other cellular processes function.

Myxobacteria were first observed in 1892 by the eminent biologist Roland Thaxter [75, 175]. Although his research interests typically focussed on fungi, he also did work on plants and bacteria. His work was characterised by detailed study and observation and the use of then state of the art techniques for micro-organism study, which are now common place in research labs. He observed myxobacteria in 1888, when he first wrote about and made sketches of a new organism he had observed under the microscope. His pioneering discovery received wider attention four years later in 1892 when, over a period of twelve years, he published three seminal papers on the life-cycle and behaviour of myxobacteria (although the papers are no longer generally available due to their age). One of Thaxter's important contributions was to justify the placement of

myxobacteria into their own family; he realised their behaviour and genealogy was significantly different from other bacteria that had been discovered by then. Since then, 12 genera and approximately 40 species of myxobacteria have been discovered and described [134].

2.2 Biology

Most strains of myxobacteria are soil dwelling; they are a very common organism which can be found in any soil sample collected anywhere in the world. An accurate measure of how many myxobacterial cells exist in a given amount of soil has never been determined; estimates vary from 8×10^4 cells/g to 5×10^5 cells/g [133]. A fruiting body will contain between 10^3 and 10^6 cells. The bacteria are strictly aerobic and can grow in temperatures between 6 °C and 38 °C. Vegetative cells grow optimally at 36 °C but quickly start to die once the temperature exceeds 45 °C [134].

Myxobacteria are relatively large for bacteria having a body length to width ratio of approximately 7:1; cells are generally between 3 μm to 12 μm in length and 0.7 μm to 1.2 μm in width. The rod shaped body of vegetative cells are semi-flexible [163] and bending of the cell body during movement frequently results in deviations contributing to directional changes [162]. Cells have a number of unique phenotypes amongst prokaryotic organisms and it is generally believed that all of their behaviour is controlled through a mixture of cell morphology and inter-cellular communication.

The size of myxobacteria genomes are considerably larger than the other sequenced δ-proteobacteria. The *M. xanthus* genome, for example, is approximately 9.6 mega bases (Mb) in length. This is over three times larger than the genome of *Escherichia coli* (*E. coli*), another well studied (and arguably better well-known) model organism. The unusually large genome size appears to be mainly due to gene duplication and divergence [48]. Genes were not duplicated at random: genes for cell-cell signalling and small-molecule sensing were amplified preferentially suggesting that these specific duplications enabled the evolution of the complex signalling required for a multicellular lifestyle.

M. xanthus is able to feed efficiently on the proteins of a wide variety of bacteria

and yeasts. Cells can synthesise all of the amino acids but lack *ilvC* and *ilvD* genes, which are necessary for the biosynthesis of leucine, isoleucine and valine. Predation seems to be used as alternative to biosynthesis as the required amino acids can usually be extracted from the proteins *M. xanthus* obtains from its prey [22].

2.3 Distribution in nature

The myxobacteria are ubiquitous in most environments, climate zones and altitudes, predominantly in top soils. Cells favour dry environments for growth, but they are also present in freshwater (probably due to soil getting washed off the land) and have also been isolated from sea water [66]. They seem to favour warmer environments and grow most efficiently at 30 °C [134] and are commonly found in environments ranging from pH 5 to pH 8 although they can be found in more extreme pH, indicating their high tolerance and adaptability to their surroundings. Common habitats include decaying plant matter such as rotting wood and the dung of herbivorous animals (laboratories typically favour dung as the preferred source for isolating cells). The concentration of cells within an environment varies greatly from a few thousand per gram of soil to a few hundred thousand per gram, depending on location. In the United Kingdom, density varies between 2000 and 75,000 cells per gram of soil.

Myxobacteria have two phases: tan and yellow. The phases exhibit different phenotypes and are so called because of changes in cell colour. Each phase has a different response to environmental conditions. Four main types of cell were discovered to inhabit a fruiting body: yellow swarmer, yellow non-swarmer, tan swarmer and tan non-swarmer. Swarmers construct a well defined mound of cells whereas the non-swarmers do not. During lysis and sporulation, there is a significant difference between the phenotypes; approximately 10 % of the yellow swarmers will sporulate whilst the remaining 90 % lyse, whilst in the tan non-swarmers 90 % of the cells sporulate and 10 % lyse.

During vegetative growth a population will reach an equilibrium where yellow phase cells dominate with a 99 % majority to a 1 % minority of tan cells [92]. Yellow-pigmented cells form rough swarming colonies with medusoid edges. They are the

predominant phase type and the one usually described in the literature when discussing the life-cycle, morphogenesis and cell phenotypes. Tan phase cells lack pigment giving them a tan/beige appearance and fluoresce under ultraviolet (UV) light. Cells can switch between phases so a homogeneous colony in one phase can switch to a heterogeneous mix of both phases. The switching rates are unequal and cells convert from tan to yellow more prodigiously than yellow to tan hence the yellow majority in most colonies. Yellow cells convert to tan at at a rate between 10^{-2} and 10^{-3} per generation.

2.4 Inter-cellular signalling

Myxobacteria coordinate their behaviour using five major inter-cellular signals that are essential for multicellular development: Asg (A-signal), Bsg (B-signal), Csg (C-signal), Dsg (D-signal) and Esg (E-signal) [37, 79, 149]. Only the A-signal and C-signal have been characterised and their function determined. B-signal, D-signal and E-signal have been discovered but their function has not been elucidated; only the approximate timings of when they act during development [37].

A-signal. The A-signal is a mixture of amino acids and peptides and acts early in development, after 1 h to 2 h of starvation during the pre-aggregation stage [37]. Starvation appears to trigger events which result in the secretion of the diffusible A-signal. The external concentration of A-signal serves as a measure of cell numbers acting as a quorum sensing signal which ensures subsequent aggregation signals occur only when a population has reached a sufficient cell density [76, 80].

C-signal. One of the most important cell signals is C-signal, which is essential during all stages of the myxobacterial life-cycle [151]. C-signal is a 17 kDa long protein expressed by the *csgA* gene. It is a non-diffusing, membrane bound protein that cells can exchange between themselves on close contact. It is thought C-signalling occurs at cell poles requiring cells to be aligned so that C-signal transmission is efficient [85]. C-signalling in this manner may be a factor controlling spatial pattern formations. Mutants lacking *csgA* fail to sporulate but can be induced to sporulate when allowed to

develop in mixture with wild type cells [54]. Signalling mutants that are unable to produce the required developmental signal still remain able to respond to the signal when it is provided by another cell. Mutants (lacking one or more of the five signalling genes) fail to sporulate on a starvation medium. However, cells transiently regain the ability to sporulate when mixed with wild-type cells or with mutants that possess the missing signalling genes [45].

2.5 Motility

Myxobacteria are surrounded by an extra-cellular polysaccharide (EPS) which forms a matrix for a cell population when cells are in close proximity, such as in a swarm [189]. Cells move by gliding across surfaces in the EPS in large swarms with coordinated movement. The population rarely moves as a whole, rather small regions may exhibit localised behaviour. Colonies typically spread out from an aggregation centre with only the colony edges exhibiting swarming. Cells nearer the centre are much more constrained in their movement. Movement is controlled by the adventurous (A) and the social (S) motility systems [101, 147] which at any one time are active at opposing poles. S-motility is coordinated at the leading pole; cells extend type IV pili which can adhere to the surface of other bacteria or polysaccharides, and upon retraction the cell is pulled forward. FrzS is a component of the S-motility engine. A-motility is less well understood. It is generally believed that cells extrude a slime from the lagging end which expands and generates a propulsive force to push cells forward [156, 196]. Although it is difficult to accurately measure the force produced by slime extrusion, Wolgemuth estimated a force of 50 pN to 150 pN [195]. It is unlikely that both motility systems are transported between cell poles, rather it is more plausible that both sets of machinery exist at both poles and are switched on or off as required [109].

The slime model of motility is not fully accepted and there is an alternative hypothesis, which has recently been proposed, suggesting cells have focal adhesion systems that mediate gliding [112]. During a reversal, the leading pole of the cell becomes the lagging pole and vice versa, so the active S-motility and A-motility engines swap poles.

2.6 Pili and fibrils

Myxobacteria have two extracellular appendages in the form of pili and fibrils [7], which are crucial to their motility systems and multicellular behaviour.

Pili. Cells have polar located pili (Figure 2.1(a)) which extend up to 10 μm or approximately one cell length. Pili play an important role in the S-motility system [74]. Cells can extend pili and grab onto the other cells. Upon retraction of the pili, the two cells are pulled closer together.

Fibrils. In addition to pili, cells also extend much longer hair-like fibrils (see Figure 2.1(b)) up to 50 μm in length with a diameter between 15 nm and 30 nm. Fibrils are more flaccid than pili and are located all over the cell body. They are formed from polysaccharide produced by the cell when cell density is very high and form an interconnecting coating around cells. Fibrils appear to be important in coordinating cell cohesion and motility though there is evidence to suggest that they are not essential and their exact functionality has yet to be determined [37].

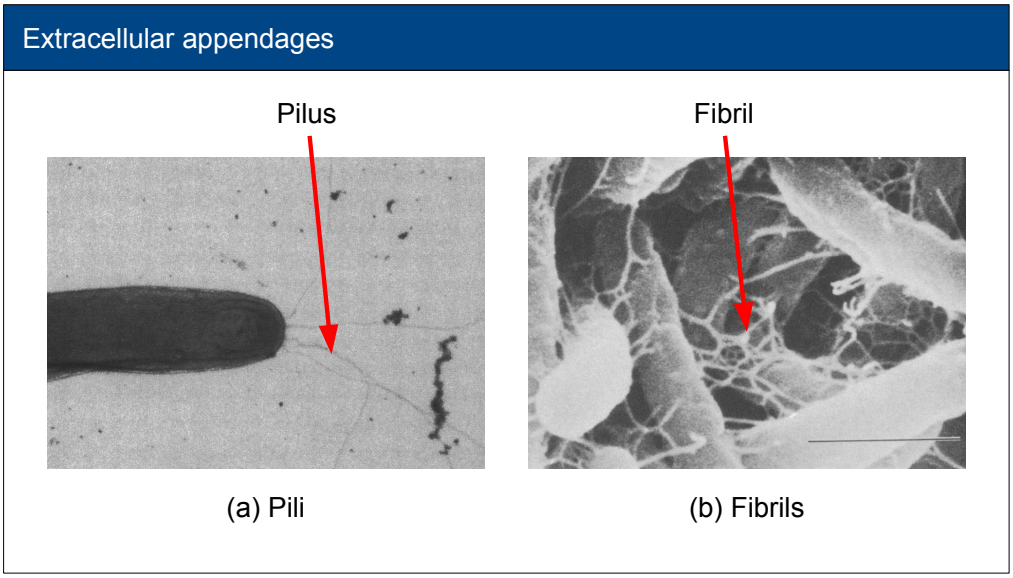


Figure 2.1: Electron micrographs of extracellular appendages. (a) Pili are relatively short appendages up to 10 μm long which are located at the cell poles. (b) Fibrils are much longer, up to 50 μm in length located all over the cell body forming a polysaccharide matrix around the cell. Images adapted from (74) and (7) respectively.

2.7 The life-cycle of myxobacteria

Myxobacteria are characterised by a complex life-cycle comprising a number of phases of behaviour which are summarised in Figure 2.2. Cells maintain a very well ordered structure between themselves regardless of the life-cycle phase [126]. It is important to understand the function of each phase and how cells transition from one stage to another so each stage is explained in more detail.

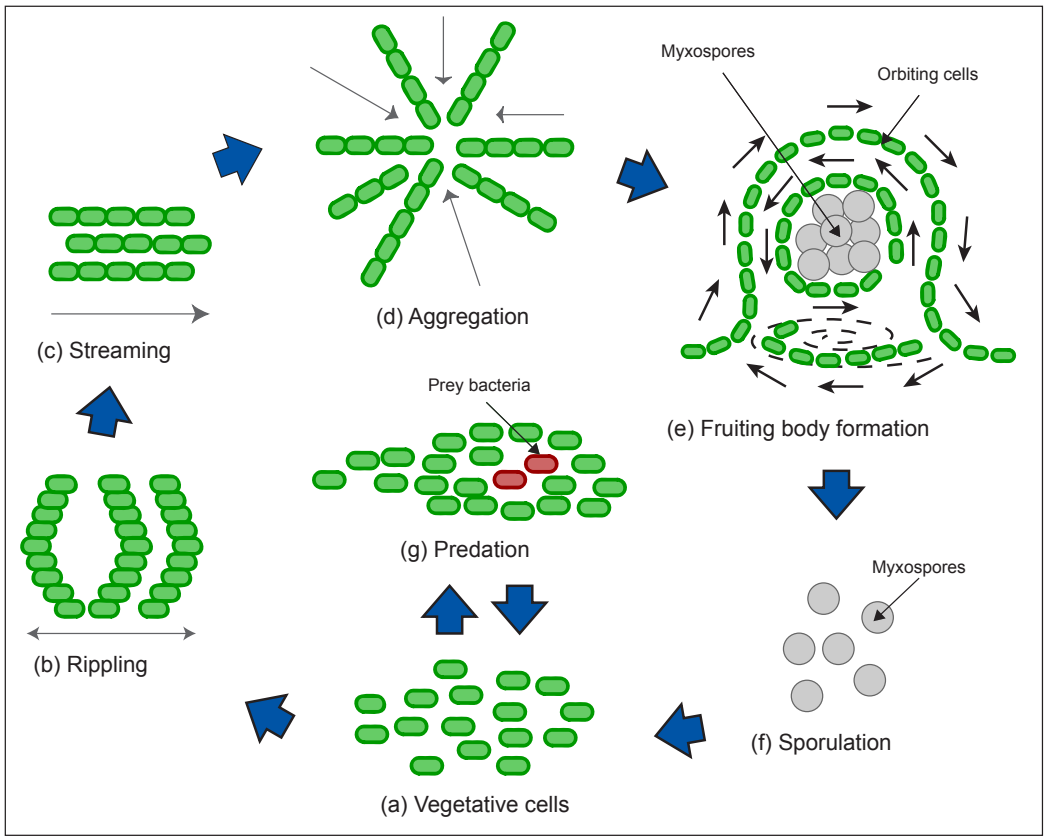


Figure 2.2: The life-cycle of myxobacteria. The life-cycle comprises a number of disparate stages which cells transition through. In a normal environment, cells are in a vegetative state ((a)) where they can predate on other bacteria ((g)). Stages (b) to (f) are observed during starvation periods.

2.7.1 Vegetative cells

In a nutrient rich environment, myxobacteria revert to a vegetative state. Cells glide in fairly straight line paths in the direction of their long axis, reversing direction approximately every ten minutes [71]. During this phase cells can swarm and predate on other

bacteria for nutrients (see Figure 2.2(g)).

Cells transition out of the vegetative phase in response to starvation and proceed through phases Figure 2.2(b) to Figure 2.2(f), which appear to act as a defence mechanism to stop the colony dying. It should be noted that all of these observations were made studying cells under laboratory conditions since it is impossible to monitor the coordinated behaviour of cells in a natural environment.

2.7.2 Rippling

Rippling (see Figure 2.2(b)) is one of the most well studied and characterised phases and paradoxically one of the most puzzling. The conditions under which rippling is triggered are known, but whether it serves any practical function within the life-cycle remains elusive. It requires energy and a signalling mechanism to coordinate behaviour, which would have presumably been removed through evolution if it served no useful purpose. Recently Berlemann et al. [11] reported that rippling may serve as a localisation function to keep cells close to nutrient sources; however, this has not been shown conclusively.

Rippling can be triggered if a colony of myxobacteria cells have been completely starved of nutrients on a minimal medium. Cells adjust their reversal frequencies and coordinate into dense bands of synchronised cells (see Figure 2.3). It was initially thought that rippling was purely a consequence of simple Newtonian physics and cells were colliding and being forced to reverse by opposing cells. Cell tracking disproved this idea by showing that cells also reverse direction in ripple troughs (although less frequently than in ripple crests) indicating that a signalling mechanism is also required [141].

Rippling is thought to be controlled by direct end-to-end contact of waves of cells that collide into each other. Cells exchange C-signal at their poles so colliding cells stimulate C-signalling [84, 188]. The level of C-signal within a cell acts as a trigger mechanism to cause reversal and alter the cell's reversal frequency, allowing it to adjust its movement to match the cells around it.

A refractory phase occurs just after reversal where myxobacteria become largely

insensitive to C-signal. Cells that are in a refractory state move through the wave of oncoming cells so that the outgoing waves are populated by cells from both incoming waves. This effect refocuses the wave at each collision, countering the dispersive effect of randomising diffusive motion of the cells [65].

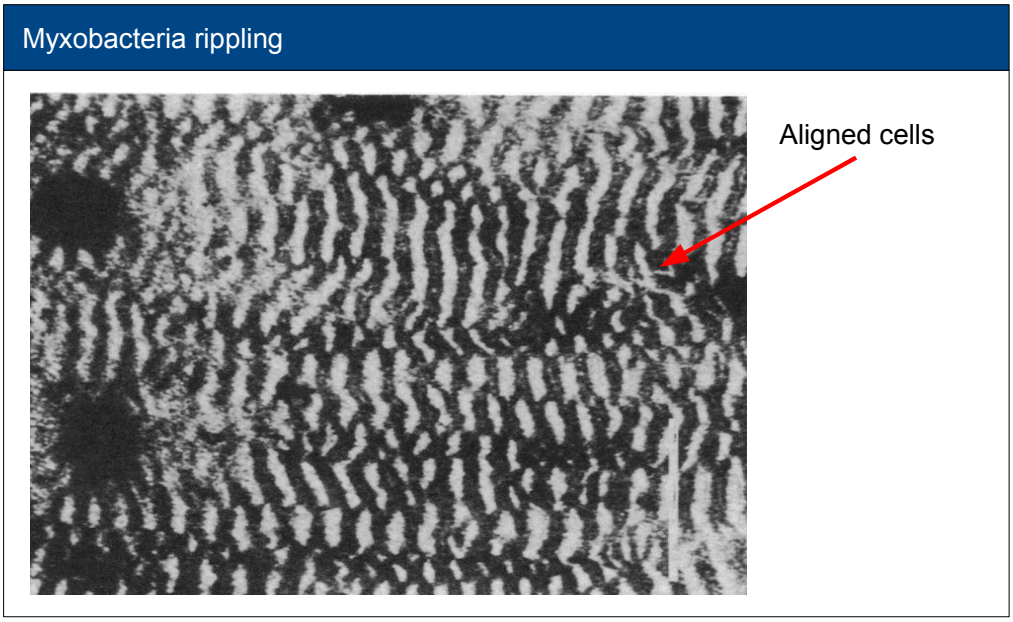


Figure 2.3: Rippling cells. Cells have formed into dense, aligned bands with synchronised reversal cycles. Dark regions are areas of high cell density. Image adapted from (150).

Once cells have begun to ripple, the synchronised patterns are highly stable and robust; ripples can be made to persist for several days [188]. During this time the ripple speed varies between 1.65 ± 0.01 μm/min and 2.66 ± 0.05 μm/min [71].

2.7.3 Streaming

As C-signal continues to rise within a cell, its reversal frequency will decrease and rippling will eventually cease. Cells will start to preferentially move in one direction and the S-motility system will cause the aligned cells to form pole to pole chains of cells which move cohesively (see Figure 2.4).

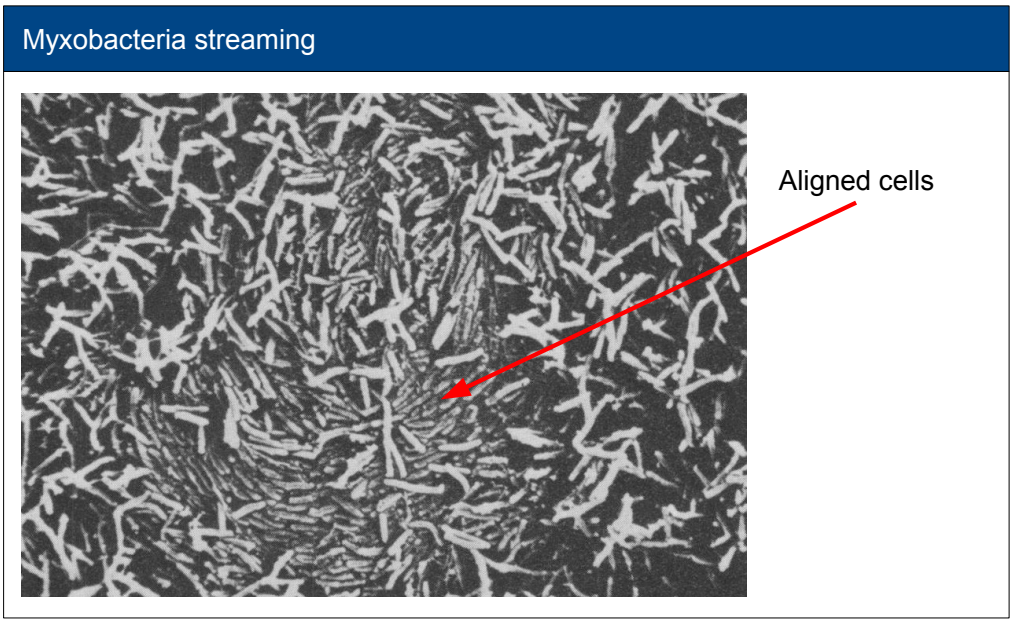


Figure 2.4: Streaming cells. Cells have aligned with each other to form streams of pole to pole cells. The cells move cohesively as a swarm. Images adapted from (118).

2.7.4 Fruiting body formation

The culmination of the myxobacterial life-cycle is the formation of a *fruiting body*. A fruiting body is a large multicellular three-dimensional aggregate containing approximately 100,000 cells. The aggregate is so large it can be seen on a plate surface without the aid of a microscope. The most simple fruiting bodies are mounds of soft or hardened slime in which the myxospores are embedded. A more advanced version develops a resistant wall around the slime forming a second protective coating. This structure is called a *sporangiole*. Fruiting bodies tend to occur in clusters with only a few species developing isolated fruits. Myxobacteria are a slime mould organism, but unlike other slime mould organisms they do not emit diffusing, chemotactic signals, such as cyclic adenosine monophosphate (cAMP) in the case of *Dictyostelium discoideum* [124, 187].

The development of a fruiting body is an ordered process going through a number of sequential stages itself [192]: growth, aggregation, mound formation, autolysis and finally myxospore induction. Figure 2.5 shows time lapse photos of the formation of a fruiting body using an electron microscope. The process takes approximately 72 h to go from starved cells to the formation of the fruiting body or fruit.

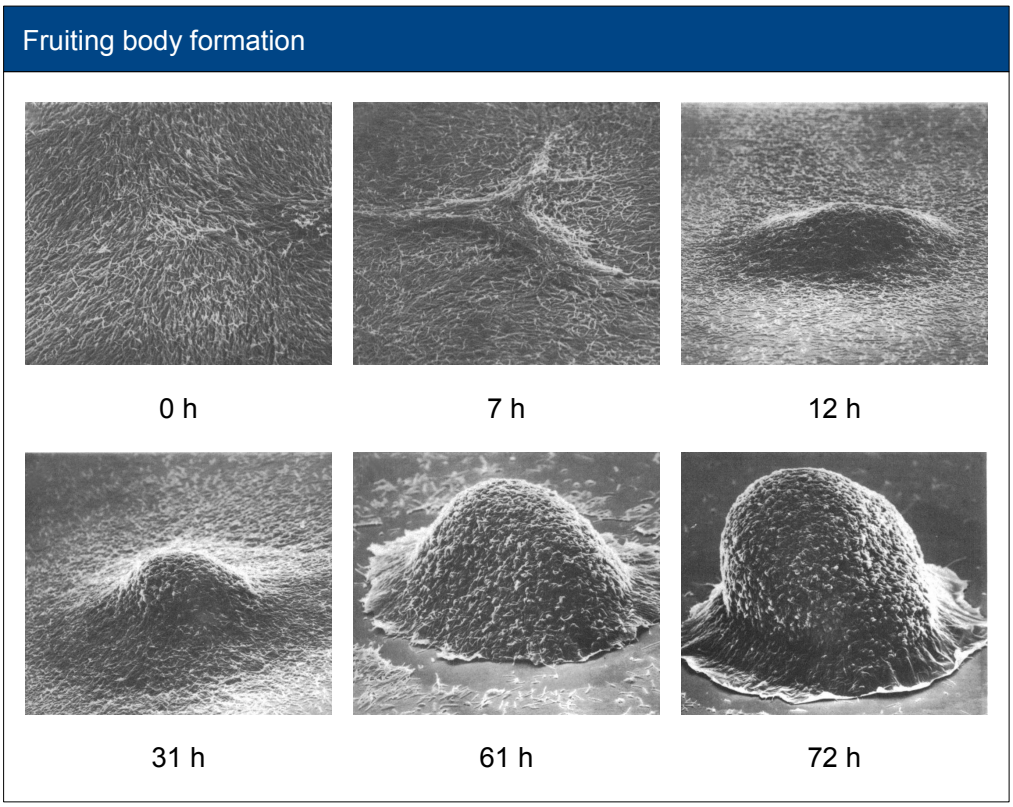


Figure 2.5: Fruiting body formation from vegetative cells to mature fruit. Images adapted from (90).

C-signal plays an important role in fruiting [86]. At low cell densities, cells begin to exchange C-signal causing a rise in the C-signal which triggers aggregation. Both A-signal and C-signal work in tandem to monitor cell density. If cell density is low, A-signal will be low so cells will continue to grow until the density is high enough that they can begin C-signalling. C-signal is positively up-regulated by its own synthesis so that there is a rapid rise in the amount of C-signal within a cell up until the point of sporulation [86]. Fruiting and sporulation can be induced with the addition of 1 nM of pure C-signal into a cell population [86]. C-signal appears to instruct cells to move at high speed with a lower reversal frequency aiding in fruit formation [70].

Fruiting appears to depend on the timing of two important conditions: the coordinated movement of individual cells so that a sufficiently high density is reached at a particular point to start the fruit and cells must also sporulate. If the two events are unsynchronised, cells can prematurely sporulate before they have reached the centre

of the fruiting body [104]. Before the fruit forms (see 0 h in Figure 2.5), cells will be organised into ordered domains which are oriented towards a general centre point [90] and swarm in spiral patterns [118]. Cells within a colony are tightly packed so it is inevitable that streams will converge and collide. Cells have little room to manoeuvre and the high levels of C-signal reduce reversals causing cells to push together and form an aggregation centre. Other cells in the vicinity that move into the aggregate will also end up getting trapped so the aggregate rapidly increases in size.

During the early stages of fruiting development (see the 7 h stage in Figure 2.5), several small aggregation centres can form in an area. After 12 h to 24 h of starvation, the streams of spiralling cells begin to form mounds with a diameter approximately equal to the starting aggregation centre. One aggregation spot will eventually come to dominate and grow bigger than the others (this would appear to be a stochastic effect and not predetermined). The smaller aggregation centres will be subsumed and a more circular aggregation spot will form. Cells at the bottom of the aggregate form the base of a stalk. New cells entering at the bottom push the stalk upwards in a manner similar to a self-erecting tower crane[1] causing it to grow and elevate the main part of the body. From 12 h to 72 h cells will continue to influx and build up on top of the aggregate until eventually a hemispherical fruit has formed.

The significance of spiralling is not yet fully understood. Although O'Connor and Zusman [118] observed spiral patterns in their studies of fruiting body formation, Kuner and Kaiser [90] did not. The spiralling behaviour has not been extensively researched since these articles were published. Recently Curtis et al. [31] showed that cells form tightly cohesive sheets which appear to collide with each other forcing one of the sheets over the other. This allows cells to build up the stalk in layers.

Massive autolysis occurs during fruiting body formation and suggests regulated autolysis is an integral part of development [192]. Sporulation appears to be dependent on lysis products, so lysis is a necessary step to induce myxospore formation. It may provide both a regulatory signal as well as necessary nutrients for surviving cells. Cells

[1] Readers may wish to visit `http://science.howstuffworks.com/tower-crane4.htm` for an explanation of how a tower crane is erected.

behave altruistically with the majority of cells dying to allow the few remaining cells to sporulate. The selection mechanism cells use to decide which ones will lyse and which will sporulate has yet to be elucidated.

Lysis is an irreversible reaction triggered by starvation although it does not occur when cells are starved in liquid suspension e. g. distilled water. Once cells have committed to lysis, restoration of nutrients does not cause them to revert back to a vegetative state. Approximately 25 % of cells in a starved population will continue to lyse even when placed in a rich medium [192]. This chain of events appears to form part of a commitment point. Cells have the ability to commit to a particular course of action rather than simply responding explicitly to every external stimulus.

Fruiting body formation may, in part, be due to a need of a colony to maintain an optimal swarm size at germination so that it can continue to prey effectively [140]. At high cell densities there is cooperative hydrolysis of casein resulting in increased growth rates. Yellow cells on casein give rise to colonies after 10 to 12 days. Tan phase cells do not form colonies for approximately 30 days. Yellow phase cells grow faster than tan phase at densities lower than 10^6 cells/droplet; however, as densities increase above 10^6 cells/droplet, the doubling times of both strains become very similar. Growth of both strains depends on the initial inoculum size. The higher the initial cell density, the shorter the doubling time. Tan phase cells appear not to be able to use nutrients as effectively as yellow phase cells at low concentrations.

Although a large proportion of cells sporulate, a proportion of cells remain motile and swarm around the periphery of the fruit. These are of interest because they appear to constitute a third type of cell [120]. Although rod shaped and exhibit similar phenotypes to vegetative cells, they express different marker proteins indicating they are functionally distinct from vegetative cells. The primary activities of vegetative cells are growth and division; however, the peripheral rods have never been observed to divide. There is a direction correlation between nutrient levels in the medium and the proportion of peripheral rods around a fruiting body. There is evidence to suggest that if nutrient levels are only slightly below what a cell requires, rather than committing fully to fruiting, which is lengthy and energy intensive process, cells persist in

the peripheral rod state so that if nutrient levels suddenly rise, they can switch back to vegetative cells and take advantage of the nutrients [119]. If nutrients suddenly disappear then the peripheral rods can commit to fruiting as normal. This mechanism serves to stop cells making hasty decisions about whether to fruit or not and ensures long term survival.

Long term starvation affects cell motility, making aggregation and fruiting more favourable. Cell speed increases from approximately 1.65 ± 0.01 μm/min to 2.66 ± 0.05 μm/min and reversal frequency decreases from 0.104 ± 0.011 reversals/min to 0.045 ± 0.002 reversals/min (see Table 1 in [71]). Cells which are not reversing are more likely to collide with other cells and start aggregating as they are not changing course and are less capable of escaping from a "traffic jam" since they cannot reverse away from obstacles.

Hodgkin and Kaiser looked at mutant cells where the two motility systems were switched on (+) or off (-). A-S+ mutants fruit well, whereas A+S- mutants fruit poorly or not at all. Cultures of A-S- strains are more cohesive and have a greater tendency to clump than A+S- strains. S motility appears to facilitate fruiting by increasing the aggregation of cells [58].

The complexity of fruiting bodies varies immensely, from a single sporangiole on a single stalk to a branched stalk with each branch bearing one or more sporangioles itself. The stalk can be composed entirely of slime, cells or a mixture of the two. There is a great deal of variety amongst the physical characteristics of each fruit. Figure 2.6 illustrates a small selection of different fruiting body formations.

2.7.5 Sporulation

Within a fruiting body, a percentage of cells will undergo morphogenesis and turn into dormant resting cells called *myxospores*. During the fruiting formation the majority of cells will undergo morphogenesis going from rods to spherical myxospores between 30 h and 72 h after starvation [90]. Rod shaped cells persist at the periphery of the fruit whilst the centre is much less motile.

Myxospore formation includes significant restructuring of cell shape with vegeta-

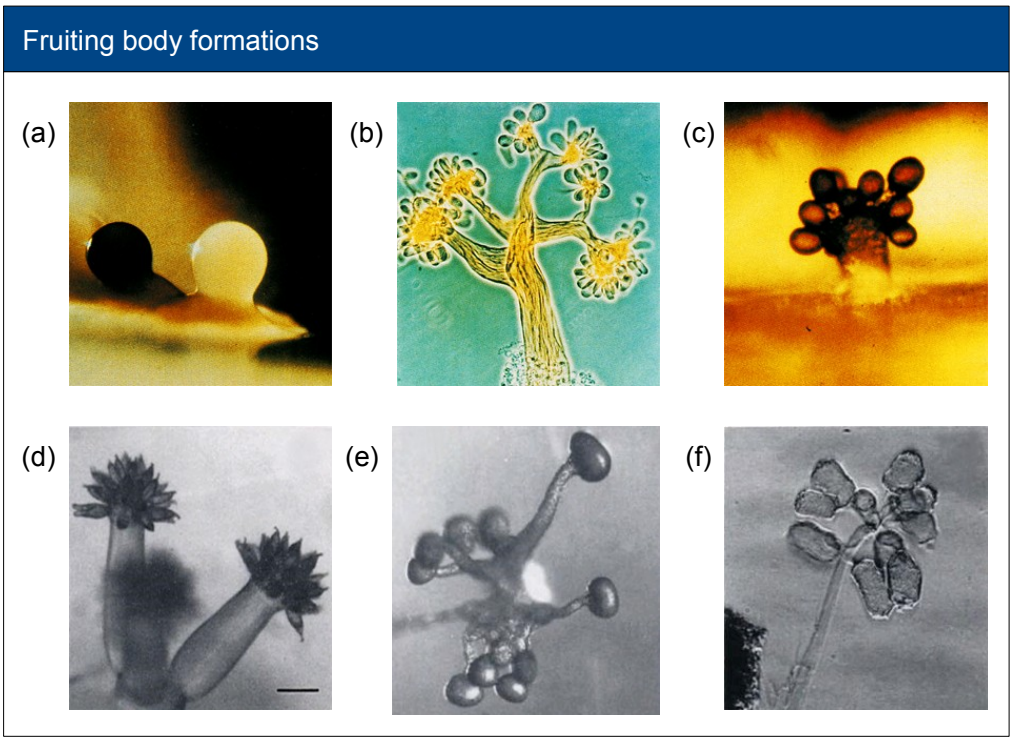


Figure 2.6: Examples of the variety of fruiting body formation in different species of myxobacteria. (a) *Myxococcus fulvus*. (b) *Stigmatella aurantiaca*. (c) *Chondromyces crocatus*. (d) *Chondromyces apiculatus*. (e) *Stigmatella erecta*. (f) *Chondromyces pediculatus*. Images adapted from (37) and (134).

tive cells going from rod shapes approximately 5×0.75 μm in size to nearly spherical with a 1.2 μm diameter [190]. The spore coat consists of multiple thin layers joined to form a thick coating around the cell.

Myxospores are desiccation resistant whereas vegetative cells die as soon as they become dry. Desiccation resistance is an important property of the spores because myxobacteria often live within top soil which can experience freezing and very harsh dry conditions. They can survive up to 16 years in laboratory conditions once desiccated and there is evidence that wild spores can survive even longer in their natural soil habitat [190]. Myxospores show resistance to heat and light, withstanding temperatures up to 60 °C for approximately 30 min in an aqueous environment and up to 90 °C once desiccated (though only for 15 min). These temperatures are well within the range of temperatures myxobacterial cells might experience in either top soil, fresh water or the sea ensuring survival.

Myxospore development can be induced with glycerol. This quickly induces a suspension in the synthesis of DNA, RNA, protein and phospholipid. There is an approximate 200 % increase in polysaccharide with approximately 50 % in the soluble portion of the cell [190]. Although glycerol induces spores, they do not form in the same way as wild-type cells. Glycerol causes rapid formation of spores in under 2 h [136, 152]; this is so fast that cells do not have time to form fruiting bodies. After sporulation during germination, the transition from spore to vegetative cell is also atypical.

Once nutrients become available again, spores will germinate and cells will revert back to their vegetative state and begin their life-cycle once more. The spore is a structural coating around the compressed cell inside. During germination a spore becomes less optically refractile and the cell will begin to elongate and push against the capsule wall. Cells appear to contract slightly before expansion which is thought to fully separate the cell from the capsule so it is easier to grow [182]. Cells appear to use enzymes to partially break down the capsule wall so that one of the cell poles can penetrate through and the cell can move out of the capsule, leaving behind an empty spore shell [183].

2.8 Population level cheating

Cooperative behaviour of myxobacteria allows some phenotypes lost by mutation to be restored by the artificial introduction of deficient proteins such as C-signal [86] or intermixing mutant cells with wild type cells. This altruistic behaviour of cells can be exploited and instances of cell cheaters within a population have been found [177].

Social cooperation relies on every member of the population working together and altruistically working towards the common good of the population. In the animal world there are species which exploit the "good will" of others in both their own and other species. Exploitation also occurs at the microbial level which is surprising given the relative simplicity of such organisms. Myxobacteria are a prime exponent of this. Cells engage in multiple social activities including feeding, predation, movement and fruiting (for a general review of myxobacterial behaviour see [37]). There are varying degrees of social exploitation [40]. Individuals can be facultatively selfish where

they exploit others but show nepotism towards their close kin and are only altruistic amongst their own kind or they can be obligate cheaters which are defective in some behaviour and must exploit others for their own evolution.

2.9 Novel uses of myxobacteria

As well as exhibiting interesting multicellular behaviour, myxobacteria are increasingly being used in novel ways to address real world problems.

2.9.1 Cancer treatment

Myxobacteria cells synthesise antibiotics to lyse prey organisms for nutrients [139]. Recently, there has been research to determine whether any of these antibiotics have a medicinal application in humans or, if they do not, whether cells can be made to synthesise useful compounds in addition to what they produce naturally. One species of myxobacteria, *Sorangium cellulosum*, produces a class of metabolite called epothilones [135]. These are are a novel class of antineoplastic agents[2] possessing anti-tubulin activity. They suppress tumour development by decreasing tubulin dissociation and arresting the cell cycle, preventing cell division. Epothilones are non-specific in their target and can therefore interfere with normal cell development however they may prove to be an effective drug against cancer.

Tubulysin A (*tubA*) is a natural product isolated from the *Archangium geophyra* species of myxobacteria [83] that has been shown to depolymerise microtubules and induce mitotic arrest so it shows potential as an anticancer and antiangiogenic agent.

2.9.2 Stone restoration

The modern, industrial age has brought about increased pollution, which has had a detrimental effect on buildings. In the United Kingdom and many other countries, a great number of old buildings were made with limestone fascias. Architectural and sculptural stone undergoes deterioration due to physical, chemical, biological and weathe-

[2] Antineoplastics are drugs that inhibit and combat the development of the abnormal growth of cell clones, which commonly lead to tumours.

ring effects which are threatening to destroy many historic buildings. Previous efforts to protect stonework typically relied on coating the stone work with a protective substance, but none of these have proved satisfactory and bring their own problems. Myxobacteria have found a niche use in building conservation [137, 186]. By spreading cells over a stone surface with an appropriate media, biofilms will precipitate calcium carbonate plugging holes in stone work. This method is effective because cell growth is limited to how long the nutrient source lasts. Cells can be killed before the bacteria actually starts to cause damage from being on the stone surface for prolonged periods. Rodriguez-Navarro et al. [137] found that between five and ten days produced optimal calcium carbonate deposition.

Chapter 3

Computation models of biological systems

3.1 Introduction

There are three primary uses of models in science [52]:

1. Understanding how a real, physical system or theory behaves.
2. Predicting the future or some unknown state.
3. Controlling and manipulating a system to produce a desirable response.

Understanding a system is related to the type of model being constructed. Haefner [52] describes the different ways models are used. These are summarised in Table 3.1, which shows the relationship between model use and the knowledge that can be obtained from it. Given the excitation and response of a system, the data can be used to construct a system to match the response and understand how a system works. This is particularly relevant in a biological context where the system itself often remains a *black box* and its internal function must be determined from data measurements of its responses to external stimuli such as mutation or changes in nutrients;.

Table 3.2 lists the meaning of some terms commonly used in the context of modelling, which are used in subsequent chapters.

Table 3.1: Uses of models. Summarised from Haefner (52).

Type of problem	Given	To find	Use of model
Synthesis	Excitation, Response	System	Understanding
Analysis	Excitation, System	Response	Prediction
Instrumentation	System, Response	Excitation	Control

Table 3.2: Modelling terminology. The definition of some key terms used when describing models and simulation.

Term	Description
Analytical model	a mathematical model whose solution is obtained purely by mathematical argument rather than through simulation or numerical approximation.
Dynamic model	a mathematical model that describes how quantities change over time.
Mathematical model	a set of equations (usually ODE or PDE) to describe a system.
Simulation	a run of a computer program that numerically solves a model.
Simulation model	a mathematical model whose solution is obtained by numerical approximation.
System	a collection of objects and relations between objects.
System state	a set of numerical values representing all system objects at a given time.

3.2 The modelling process

To aid in development of models, a semi-formal set of rules is often adopted during design to try and formalise and order the process [53]. At its most basic level, modelling a system is about *producing* a model, *implementing* it in a formal language either purely mathematically or computationally, *deriving predictions* from the model and finally *evaluating* the predictions to determine if the model is fit for purpose.

The process of modelling can be formalised into a number of tasks which make the modelling process more robust and increase the likelihood of the model being successful.

Objectives. The purpose of the model must be clearly stated and the scope of the problem understood.

- Can the system being modelled be described succinctly?
- What are the major questions the model should answer?
- How accurate does the model need to be? How is the validity of the model being measured?
- How will the model be analysed? Does it provide all of the necessary data?

Hypotheses. The objectives and all knowledge of the system needs to be translated into a list of hypotheses with a clear statement of what they mean.

Mathematical formulation. Hypotheses are typically qualitative in nature and need to be converted into quantitative relations that can be formulated with mathematical equations and evaluated.

Verification. Many mathematical models do not have analytical solutions and must be solved numerically. Since this is an approximation of the real system, the numerical method must be verified to ensure it is giving correct results.

Calibration. A model of a biological system may involve many variables so appropriate initial conditions for the system must be found. The system may involve multiple parameters which must be determined either from experimental data or by parameter estimation.

Evaluation. The results should be validated and compared with the objectives to determine the quality of the data.

The classic model requires that a single model is created and verified. It can only be discarded after exhaustive testing, meaning that the development process can be long if the model requires a number of changes. An alternative approach to speed up development is to develop multiple hypotheses and test them simultaneously. The modelling process described here was applied during the design of the systems in Chapter 6, Chapter 7 and Chapter 8.

3.3 Modelling in a biological context

Mathematical modelling of biological systems offers a methodology to discover organisational mechanisms used by interacting cell systems [32]. This is important for understanding how individual organisms behave and how biological patterns manifest themselves. Analysis of single gene and protein functions is normally not sufficient to explain complex pattern formation.

One primary example of biological pattern formation is *morphogenesis* in multicellular organisms. This can concern the development of tissue and organ arrangements or it can focus on the life-cycle of micro-organisms and how they interact within a colony. It is the latter that forms the focus of the research outlined in this thesis.

Pattern formation is a spatio-temporal process so that to understand it, we must characterise how a system evolves over both time and space.

Theoretical models of systems can be broadly partitioned into the following classes: finite-difference equations (FDE), individual based models (IBM), ordinary differential equations (ODE) and partial differential equations (PDE). Table 3.3 summarises the different classes and how they represent space, time and state. The model classes are

Table 3.3: Approaches to modelling spatio temporal pattern formation.

	Model	Space	Time	State
1.	Difference equations	Continuous	Discrete	Continuous
2.	Interacting particle systems	Discrete	Continuous	Discrete
3.	Cellular Automata	Discrete	Discrete	Discrete
4.	Coupled ODE	Discrete	Continuous	Continuous
5.	PDE	Continuous	Continuous	Continuous
6.	Set of rules	Continuous	Discrete	Continuous

linked to each other and through refinement and specialisation of a model class, it is possible to transition one class of model to another. One modelling class is not superior to another, either quantifiably or qualitatively. Each has a number of positive and negative characteristics and the choice is dependent on the problem being considered and the essential features it must possess.

Models dealing with spatial pattern formation are implicitly reliant on an accurate depiction of space if they are to capture intricate and complex patterns of movement. Therefore, the important classes are those that allow space to be simulated in a continuous domain. The clear choices are therefore PDE systems and rule based models. A PDE system can describe all variables of a system in a continuous domain, but it suffers from a lack of scalability. Each entity in a model will require one or more equations to describe it, so a system modelling thousands of entities/agents/cells would need to solve a very large number of equations. The system is complex due to both the number of equations to be solved and simplifying assumptions required to achieve analytical or computational tractability [125]. Differentiation amongst entities in a non homogeneous system can also be problematic as a PDE is not very efficient at describing the internal state: many systems have a number of binary states, for example, that describe particular properties of the entity such as size or whether a gene system is switched on or off. A differential equation cannot elegantly describe states in this manner since it is designed to describe how a quantity is changing over time and does not elegantly capture event logic.

Individual based models allow physical space to be described continuously whilst offering a discrete representation of time and state. Discrete time and state can make model calculations faster since they can be computed at regular intervals, missing out intermediate values that would need to be computed in a continuous paradigm (with an appropriate PDE or ODE solver). Although there is some loss in accuracy, this does not mean that individual based models are incapable of properly describing a system. The Nyquist-Shannon sampling theorem states that:

> If a function $x(t)$ contains no frequencies higher than B Hz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart.

Therefore, if a system is sampled sufficiently often, a discrete system can describe an analogue continuous system. Individual models must choose an update interval that is smaller than the time interval between fluctuations in the fastest component of the model to capture the behaviour of the system.

Scale is an important consideration when deciding how to model a process. If the time scale is too short, a variable may behave as a constant leading to a false interpretation of the model. If the time scale is too long, a lot of time may be spent unnecessarily simulating portions of the model that yield no useful results.

3.4 Cellular automata

From Table 3.3 it should be apparent that cellular automata and individual agent based models offer the best way to capture both space and time in a model. The individual based models offer the most realism; however, it is useful to begin with a discussion of cellular automata to introduce concepts that can be applied to individual based models.

Cellular automata (CA) are discrete dynamical systems that can be used as models of biological pattern formation [32]. They were introduced by John von Neumann [184] as means of modelling self-reproduction on a computer.

CA allow biological problems to be studied from a different perspective. One of the key insights such models have already shown is that the macroscopic behaviour of systems can be due to cells obeying a few conventional rules from classical physics to describe their motion and interaction; simple rules can give rise to very complex behaviour. This more top-down approach allows us to go from a macroscopic to a microscopic level. We can develop a model of the observed macroscopic behaviour based upon the known biology of the organism and use the results of the model to guide biological research. The CA can indicate the important factors for determining specific pattern formations, which then direct research to find the responsible biological mechanisms. This approach can be more straightforward than starting from the individual gene level and working upwards to the specific genes responsible for a particular behaviour. In the case of organisms where only a small portion of the genome has been annotated, it may prove very difficult to determine how patterns form using purely biological data.

CA are discrete models of possibly continuous systems. They can be characterised formally by a regular lattice, an interaction neighbourhood, a set of elementary states and a rule governing the transition from one state to another. Each lattice node is called a *cell* and represents an entity at that location which can be in one of the finite states. Typically the lattice represents a discretisation of space in one, two or three dimensions although this is not necessary. There is no implicit scale to the lattice so each *cell* could represent a biological cell (in which case the model is of a cell population) or perhaps molecules in a substance. At each iteration of the model, the transition rule is applied to every *cell* simultaneously in order to get the new state of the system. The homogeneous nature of the transition rule restricts certain models since each *cell* is implicitly the same with minimal variation allowed.

A CA has the following properties:

- A regular discrete lattice $L \subset \mathbb{R}^d$ which discretises a d-dimensional Euclidean space. A lattice has a connectivity b specifying the number of nodes any one node is linked to. Typically a lattice has either square ($b = 4$), triangular ($b = 3$) or hexagonal ($b = 6$) connectivity (see Figure 3.1). Boundary conditions are

specified for the edges of the lattice.

- A finite number of *cells*, with one occupying each lattice point. Note that *cell* does not necessarily refer to a biological entity; it is a name for a lattice node and can also be called a *node* if dealing with an LGCA or a *site* in stochastic process models.
- A finite set of states E representing the possible states a cell can be in.
- An interaction neighbourhood N^I defining a local neighbourhood. Each cell can use the neighbourhood to detect other cells to interact with.
- A rule R that determines the transitions between states for each cell.
- Each cell has a position $p \in L$, where p describes the lattice point location, for example on a three dimensional lattice $p = (x, y, z)$.

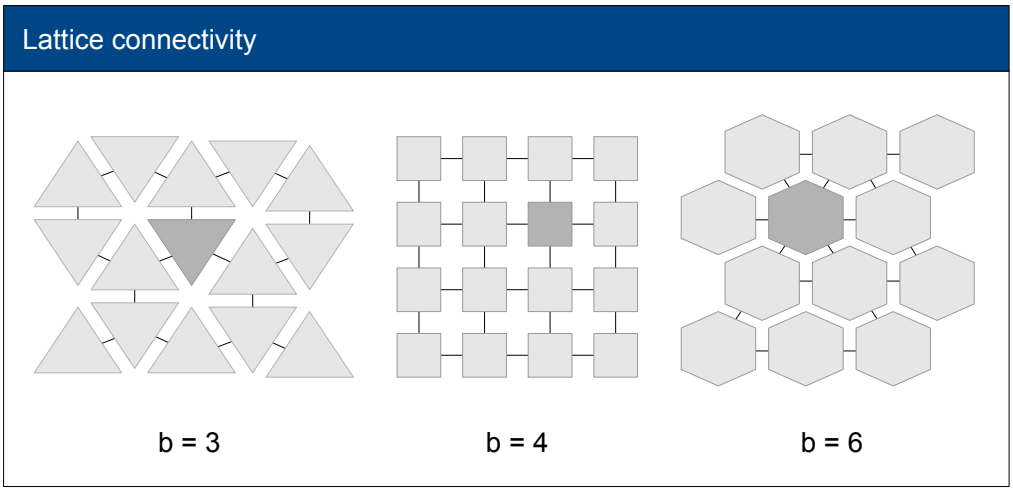


Figure 3.1: Lattice connectivity. Typical lattice formations are either triangular

Behaviour at the lattice boundary requires special attention to cope with entities moving on or off the lattice and the physical effects of the boundary itself. Four main approaches are taken to control cell behaviour at lattice boundaries.

Periodic. The boundary can be made periodic by linking opposite boundaries so the lattice volume extends infinitely in all directions. The lattice is assumed to be part of

a much larger lattice and we are only modelling a small portion of this. Rather than treating the lattice in isolation, periodic boundaries introduce stochastic effects at the edges to simulate the effect cells outside the lattice boundary have on the cells within the lattice.

Neumann. The boundary is made to be zero flux, so the value at the edge becomes an insulator.

Dirichlet. Cells have a fixed value at the boundary.

Absorbing. Cells at the edge are lost if they attempt to move over a boundary.

3.4.1 Neighbourhoods

CA models define a local neighbourhood N^i which consists of a set of locations a cell can check to determine the state of neighbouring cells. In a biological context, the neighbourhood is typically small relative to the size of the lattice to reflect that interactions are short range; distant cells have less of an influence compared to cells close by.

$$N_b^I(p_0) = \left\{ p_0 + c_i : c_i \in N_b^I, \quad i = 1, 2, \cdots, b \right\} \subseteq L, \tag{3.1}$$

where b is the connectivity specifying the number of nearest neighbours, N_b^I is a neighbourhood template specifying the nodes that will appear in the neighbourhood, p_0 is the location of the centre of the neighbourhood and c_i specifies the position of a neighbouring node relative to p_0. If the interaction neighbourhood spans a boundary, it will either be truncated so all nodes are within the lattice, or, if the lattice is periodic, the cell addresses will be computed modulo the lattice dimensions.

Although the interaction neighbourhood can be any shape and size, typically three main types of neighbourhood are used almost exclusively: Moore, von Neumann and circular. A detailed description of neighbourhoods and their implementation is given in Section 5.4.4.

The definition of a neighbourhood may optionally include p itself in the set of neighbours depending on the strictness of the neighbourhood definition being used. In the implementation of FABCell (discussed in Chapter 5), the neighbourhood centre is always added to the set of neighbours and it is up to the user to decide whether to exclude it or not.

Each cell $p \in L$ has a state $s \in E$. The state can be any arbitrary value, object or symbol as long as it is unique and distinct from the other states.

3.4.2 Lattice gas cellular automata

Lattice gas cellular automata (LGCA) are refinements of CA. They were originally developed to study the behaviour of ideal fluids and gasses but have since been used to study biological problems [2, 3, 87].

Like all CA, LGCA employ a regular, finite lattice and include a finite set of particle states, an interaction neighbourhood and local rules which determine the particles' movements and transitions between states [1]. LGCA differ from traditional CA by assuming particle motion and an exclusion principle. Each cell has a number of channels $c_i \in N_b, i = 1, 2, \cdots, b$ associated with it based on the connectivity of the lattice, which indicate the velocity of the cell at the lattice point. The velocity specifies the direction and magnitude of movement, which may include a rest (zero) velocity. More than one cell can exist at a lattice point so long as the exclusion principle is maintained. This requires that not more than one cell can occupy a channel at a same lattice point. The primary purpose of LGCA is to study the flow of molecules in gas. The CA model was therefore extended to represent an actual volume and to allow cells to move between lattice points.

Every lattice point can now be given a state representing the occupancy of each channel:

$$s(p) = (n_1(p), n_2(p), \cdots, n_b(p)) \tag{3.2}$$

$$n_i(p) = \begin{cases} 1 & \text{if cell present} \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where p is a lattice position and $n_i(p)$ indicates whether channel i of node p is occupied (1) or empty (0).

The local configuration of a node p is given by

$$\boldsymbol{s}_{N(p)} = \boldsymbol{n}_{N(p)} \tag{3.4}$$

The transition rule of an LGCA has two steps. An interaction step updates the state of each cell at each lattice site. Cells may change velocity state and appear or disappear in any number of ways as long as they do not violate the exclusion principle. In the transport step, cells move synchronously in the direction and by the distance specified by their velocity state. Synchronous transport prevents cell collisions which would violate the exclusion principle.

3.4.3 State transition

The transition rule R specifies the new state of a cell as a function of its interaction neighbourhood configuration. In a conventional CA, the rule is spatially homogeneous and does not depend on the cell position p. It is applied to every lattice point. There is no requirement for homogeneity and the rule can be dependent on position or time.

A CA can be deterministic in which case the transition rule yields a unique next state from any other state. The evolution of the system is predictable since you know *a priori* what the new state will be given a starting state.

A probabilistic CA defines a probability distribution of the next states

$$R\left(\boldsymbol{s}_{N(p)}\right) = \begin{cases} z^1 & \text{with probability } W\left(\boldsymbol{s}_{N(p)} \rightarrow z^1\right) \\ z^2 & \text{with probability } W\left(\boldsymbol{s}_{N(p)} \rightarrow z^2\right) \\ \vdots & \\ z^{\|E\|} & \text{with probability } W\left(\boldsymbol{s}_{N(p)} \rightarrow z^{\|E\|}\right) \end{cases}, \tag{3.5}$$

where $z \in E = \left\{z^1, z^2, \cdots, z^{\|E\|}\right\}$ and $W\left(\boldsymbol{s}_{N(p)} \rightarrow z^j\right)$ is a time independent transition probability specifying the likelihood of reaching state z^j from a node configuration $\boldsymbol{s}_{N(p)}$. Extending this to an LGCA the configuration state becomes $s(p,k)$.

$$n_i^I(p,k) = R_i^I\left(\boldsymbol{n}_{N(p)}(k)\right) \tag{3.6}$$

3.5 Monte Carlo methods

Statistical physics, more specifically Monte Carlo methods, play an important role in the modelling research carried out in Chapters 6, 7 and 8. This section provides a review of the Monte Carlo methods used.

Monte Carlo methods form the largest and most important class for solving statistical physics problems [116]. Statistical physics is concerned with the calculation of properties of large, condensed matter systems. Though each part of the system may itself be relatively simple and obey a straightforward set of simple equations of motion, typically the sheer number of parts being considered makes it incredibly difficult to solve all of the equations exactly. Approximation techniques are therefore required to understand a system. A classic example of the type of problem considered is a volume of 1 m^3 of oxygen at room temperature. It will contain in the order of 3×10^{22} oxygen molecules. There are currently insufficient computing resources to model every single molecule to determine their interactions. Modelling a volume of gas is a relatively simple problem and even this has a very large order of complexity making even marginally more complex systems almost impossible to solve exactly.

The macroscopic behaviour of these large systems is often predictable allowing us

to study the average effects of every component whilst being able to understand the properties of the system. We can study the gross properties of a large system in a probabilistic way even if the equations of motion are too complex.

The work in later chapters concerns a class of problems where the Hamiltonian H of the system is studied. The Hamiltonian is a measure of the total energy of a system in a particular state. The system is perturbed by a *thermal reservoir* which acts as a heat sink and heat source and attempts to push the temperature of the Hamiltonian system towards that of the reservoir. In this way the system is in a constant state of flux, being pushed from one energy state to another.

Every system is composed of a finite (albeit large) number of states which it can change between with a certain probability. The probabilities represent our knowledge of the system as it allows inference of the long term average behaviour of a system. The evolution of a system can be expressed as:

$$\frac{d}{dt}\omega_\mu = \sum \left[w_\nu(t) R\left(\nu \to \mu\right) - w_\mu\left(t\right) R\left(\mu \to \nu\right) \right] \tag{3.7}$$

$$\sum w_u\left(t\right) = 1 \tag{3.8}$$

At a given time t, the probability of a system being in state μ is ω_μ, which is the sum of the transitions into μ, $R(\nu \to \mu) - w_\mu$ minus the transitions out of that state to some other state $R(\mu \to \nu)$. The probabilities can be related to the macroscopic properties we are interested in measuring by determining the expectation of a property q across all states.

$$\langle q \rangle = \sum_\mu q \omega_\mu(t) \tag{3.9}$$

where $\langle q \rangle$ is the expectation of q. We are interested in some q which has a value p_μ in state μ.

There are two ways to relate the expectation to the real value measured during an experiment (since this is what we are ultimately interested in):

1. There are N individual copies of the system operating in parallel. If the value of q could be measured in every system instantaneously, $\langle q \rangle$ should estimate the mean value of q from all the measurements.

2. $\langle q \rangle$ can also be considered as the time average of q. If the value of q is recorded at a fixed interval for a long time, $\langle p \rangle$ should approximate the mean of all measurements, so long as the system passes through enough representative states. This is considered a less rigorous view as there is no way to quantify if and when the system will pass through enough representative states, since it could spend a disproportionate amount of time in one state, which would shift the mean. Time average interpretation is the most widely adopted approach as it best approximates the way experimental data is collected. Only equilibrium systems are considered, where the probability weights do not vary, so issues of interpretation do not arise.

3.5.1 Equilibrium

If Equation 3.7 is allowed to reach a state where the transition probabilities are equal for all μ then the transition probabilities ω_μ become invariant and the system goes into equilibrium. In a thermal equilibrium system described here, the occupation probabilities take on specific values.

Gibbs showed that for a system in thermal equilibrium with reservoir at temperature T, the equilibrium occupation probabilities are

$$p_\mu = \frac{1}{Z} e^{-E_\mu / kT}, \tag{3.10}$$

where E_μ is the energy of state μ and k is Boltzmann's constant. Z is the *partition function* which acts as a normalising constant.

$$Z = \sum_\mu e^{-\beta E_\mu} \tag{3.11}$$

Knowing how Z varies allows us to understand the macroscopic behaviour of a system. For example, the internal energy (U), specific heat (C) and entropy (S) can all

be derived from the partition function.

$$\langle Q \rangle = \frac{1}{Z} \sum_{\mu} Q_{\mu} e^{-E_{\mu}/kT} \tag{3.12}$$

$$U = -\frac{1}{Z} \sum_{\mu} E_{\mu} e^{-\beta E_{\mu}} = -\frac{\partial log Z}{\partial \beta} \tag{3.13}$$

$$C = \frac{\partial U}{\partial T} = -k\beta^2 \frac{\partial^2 log Z}{\partial \beta^2} \tag{3.14}$$

$$C = T\frac{\partial S}{\partial T} = -\beta \frac{\partial S}{\partial \beta} \tag{3.15}$$

$$S = \int C \, d\beta = -k\beta \frac{\partial}{\partial \beta} log Z + k log Z \tag{3.16}$$

3.5.2 Fluctuation

It is desirable to understand the behaviour of the system over time and how properties are changing. One useful measure is the standard deviation of a property which gives a measure of the variation over time. The standard deviation of internal energy U can be expressed as

$$U = \langle E \rangle \tag{3.17}$$

$$\langle (E - \langle E \rangle)^2 \rangle = \langle E_2 \rangle - \langle E \rangle^2 \tag{3.18}$$

$$\langle E^2 \rangle = \frac{1}{Z} \sum_{\mu} E_{\mu}^2 e^{-\beta E_{\mu}} = \frac{1}{Z} \frac{d^2}{\beta^2} Z \tag{3.19}$$

$$\langle E^2 \rangle - \langle E \rangle^2 = \frac{C}{k\beta^2} \tag{3.20}$$

3.5.3 Isling model

The models presented here and subsequent chapters are derivatives of the Isling model, so a brief review of how it functions is presented here. It forms a concrete example of theory discussed so far.

The Isling model is a model of a magnet which looks at the magnetic dipole movements of individual atomic spins within a bulk material. A material is modelled as a simple geometric shape which is discretised using a regular lattice. An atom is placed at each node forming the structure of the material. Each atom can have one or more spin states (typically represented as a numerical value). Each atom has two states, ± 1, reflecting that a magnet has two oppositely charged poles[1] and each atom will be one of the two charge states. The spins interact with bulk spins locally aligning to form the dipoles.

The Hamiltonian of an Isling model includes terms proportional to the product of neighbouring spins $s_i s_j$, where i and j are two neighbouring atoms, so it is favourable for spins to align with each other.

$$H = -J \sum_{i} \sum_{j \in N(i)} s_i s_j, \tag{3.21}$$

where J is a measure of the interaction strength of spins and $N(i)$ is a list of the neighbouring atoms of atom i. Favourable changes in the system energy are signified with a negative energy hence the minus sign in front of J. A key point to note here is that each property of the system can be expressed as a term in the Hamiltonian.

An Isling model has 2^N possible states for a lattice with N spins on it. The partition function can therefore be written as:

$$Z = \sum_{s_1 = \pm 1} \sum_{s_2 = \pm 1} \cdots \sum_{s_N = \pm 1} e^{-\beta H} \tag{3.22}$$

The partition function can subsequently be used to find the internal energy, entropy, free energy and specific heat.

3.5.4 Monte Carlo method

The Monte Carlo method calculates the partition function of a large Isling model in a realistic time frame. Conceptually it involves simulating random fluctuations of a system from state to state, where the system enters a given state μ at time t with a

[1] An object or system that is oppositely charged at two points or poles.

probability equal to the transitional probability $\omega_\mu(t)$ of the real system. The transition rates $R(\mu \rightarrow \nu)$ are chosen so that the equilibrium solution is equal to the Boltzmann distribution.

Only a relatively small fraction of the system states have to be sampled in order to get accurate estimates of system properties. There will be statistical errors in calculation since not every state is included. However, as discussed earlier, a long average of the system should closely approximate the real value of a parameter in the system.

The expectation of a system can be regarded as a time average over all the states a system passes through during a measurement. The possible number of states even a relatively simple system can be in is vast. However, in a laboratory experiment, systems such as gases clearly do not have the time to pass through every single state, yet the data from experiments still provides meaningful and accurate information about the properties being measured. If a real system can be measured and understood using only a fraction of the total states, then the Monte Carlo method can be considered an analogous method to examine a virtual system.

A real system containing many molecules, each with multiple states, will have an immense number of states it could be in. During the time an experiment runs for, only a small fraction of these states will be visited. A real system therefore does not sample all states equally and can be seen to sample using a Boltzmann probability distribution; it functions as an analog computer.

A Monte Carlo approach is much better than might be expected at approximating the behaviour of a real system. The probability of a state being selected is proportional to its Boltzmann weight $p_\mu = Z^{-1}e^{-E_\mu/kT}$. This is a form of importance sampling; states are not selected with equal probability reflecting that a real system will not do the same. Simply picking states at random and accepting them with a probability proportional to $e^{-E_{\mu/kT}}$ is essentially no better than picking at random since a large number of states will be rejected and a very high sampling frequency could be required to find valid states. To ensure states are selected with their Boltzmann probability a Markov process is used. A *Markov* process is a stochastic method which when given a system in one state, generates a new state in a random fashion. The transition probabilities satisfy

four primary conditions:

1. The probabilities do not change over time.

2. The probabilities are dependent only on the current and new state.

3. The process is *ergodic* so that it is possible to reach any state from any other state if it is run for long enough. This satisfies a property of the Boltzmann distribution which is that every state appears with a non-zero probability.

4. A condition of *detailed balance*. This is to ensure the system generates the Boltzmann probability distribution as it comes to equilibrium. If the transition probabilities $P(\mu \rightarrow \nu)$ are considered as a matrix $\boldsymbol{P}$ (the Markov matrix) then the probability a system is in state μ at time t, $\boldsymbol{w}(t)$ can be expressed as

$$\boldsymbol{w}(t+1) = \boldsymbol{P} \cdot \boldsymbol{w}(t) \tag{3.23}$$

With this system, an equilibrium state can be reached as $t \rightarrow \infty$ in two ways:

$$\boldsymbol{w}(\infty) = \boldsymbol{P} \cdot \boldsymbol{w}(\infty) \tag{3.24}$$

$$\boldsymbol{w}(\infty) = \boldsymbol{P}^n \cdot \boldsymbol{w}(\infty) \tag{3.25}$$

Equation 3.25 is undesirable because it forms a *limit cycle* with the system cycling through a finite number of states not necessarily following a Boltzmann probability distribution. To avoid this, the transition probabilities are constrained such that

$$p_\mu P(\mu \rightarrow \nu) = p_\nu P(\nu \rightarrow \mu) \tag{3.26}$$

On average the system should go from state μ to ν equally as often as from ν to μ. This prevents cycling since a cycle implies a transition $P(\mu \rightarrow \nu)$ occurs more often than $P(\nu \rightarrow \mu)$.

The repeated application of a Markov process yields a Markov chain of states. Each new state forms the seeding state to generate the next new state. The long term average of this process produces a chain of states which appear with their Boltzmann probability with the system tending towards equilibrium.

3.5.5 The Metropolis algorithm

The last remaining step of the method is to design a Markov process so that the appearance of each state follows a Boltzmann distribution. An *acceptance ratio* can be used to determine whether a transition from μ from ν should be accepted.

The *Metropolis* algorithm was devised by Nicolas Metropolis [105, 106] as a method of ensuring the Markov process behaves correctly. It is the most widely used algorithm for solving Monte Carlo systems and is used in the models created in Chapters 6 to 8. The basic algorithm is outlined in Algorithm 1.

Algorithm 1 Metropolis algorithm

1: Choose a set of selection probabilities $g(\mu \rightarrow \nu)$ for each possible transition
2: **repeat**
3: Choose a new state ν.
4: Accept the change of state with the following acceptance function:

$$A(\mu \rightarrow \nu) = \begin{cases} e^{(E_\mu - E_\nu)/kT} & \text{if } E_\mu - E_\nu > 0 \\ 1 & \text{otherwise} \end{cases} \tag{3.27}$$

5: **if** $A(\mu \rightarrow \nu) = 0$ **then**
6: Accept the system state change to ν
7: **else**
8: Stay in state μ
9: **end if**
10: **until** Simulation terminates

The energies of a system in thermal equilibrium tend to stay within a confined range. Fluctuations are small compared to the energy of the whole system. Systems tend to stay in a subset of states and rarely make transitions that change the energy of the system dramatically [115].

The selection probabilities between states are all chosen to be equal so that proposed changes are random and happen with an equal probability. In a system of N spins the selection probabilities would be:

$$g(\mu \rightarrow \nu) = \frac{1}{N} \tag{3.28}$$

The system must still maintain ergocity and ensure that transition probabilities follow a Boltzmann distribution.

$$\frac{P(\mu \rightarrow \nu)}{P(\nu \rightarrow \mu)} = \frac{g(\mu \rightarrow \nu)A(\mu \rightarrow \nu)}{g(\nu \rightarrow \mu)A(\nu \rightarrow \mu)} = \frac{A(\mu \rightarrow \nu)}{A(\nu \rightarrow \mu)} = e^{-(E_\nu - E_{mu})/kT} \tag{3.29}$$

In order to maximise the acceptance ratios in Equation 3.29, the larger one is always given the largest possible value it can take (in this case 1) and the other is adjusted to fit the constraint. This leads to the Metropolis algorithm:

$$A(\mu \rightarrow \nu) = \begin{cases} e^{(E_\mu - E_\nu)/kT} & \text{if } E_\mu - E_\nu > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{3.30}$$

A state transition from μ to ν always occurs when state ν has an energy lower than or equal to μ. If the energy of state ν is higher then it is accepted with the Boltzmann probability. Thus, the higher the energy of a new state ν compared to μ, the less likely it is the system will move to ν.

3.6 The Potts model

The Monte Carlo algorithm represents a general solution to studying problems in statistical mechanics. To go from this to a more specialised solution specifically focussing on biological problems requires further refinement using a Potts model. The Potts model was first described in 1952 by Renfrey Burnard Potts [128]. It is a generalisation of the Isling model (see Section 3.5.3) where spins are confined to a plane and each spin can be in one of q equally spaced state directions between 0 and 2π [199]. Formally this can be expressed as

$$\sigma_n = 2\pi n/q, \quad n = 0, 1, \cdots q - 1 \tag{3.31}$$

Figure 3.2 illustrates the generalised spin model on a lattice. Each lattice site has a

spin associated with it. Using the direction example this can be represented pictorially using direction vectors at each lattice node. The angle itself does not necessarily have to have a physical interpretation, it functions only to describe the spin state.

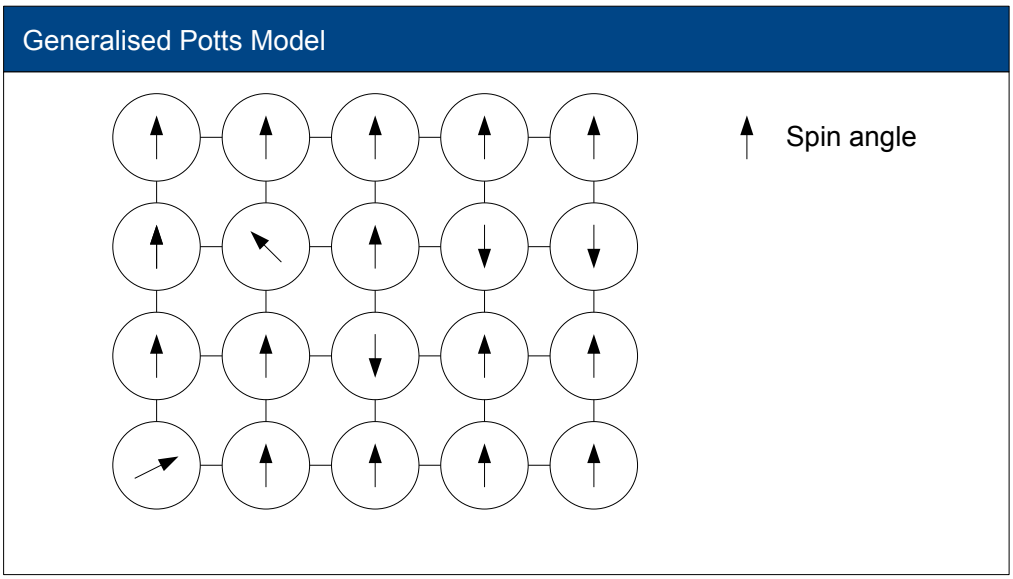


Figure 3.2: Generalised Potts model. A system consists of a lattice of spins each of which can be oriented in one of a finite number of spin states. Typically spin states are restricted to equally spaced directions in a plane.

Interaction between spins occurs between nearest neighbours and is dependent on the relative angle between spins at adjacent lattice sites. In the simplest case the Hamiltonian of the system can be described using just an interaction function.

$$H = \sum_{\langle i,j \rangle} J\left(\sigma_i, \sigma_j\right) \tag{3.32}$$

where $J\left(\sigma_i, \sigma_j\right)$ is a function which returns the correlation between σ_i and σ_j at adjacent lattice sites i and j. A simple example of the interaction function is $J\left(\sigma_i, \sigma_j\right) = \sigma_i - \sigma_j$ and J is periodic between 0 and 2π. If the states of σ are finite, J is often implemented as a two-dimensional lookup table computationally to determine the interaction quickly.

Since the Potts model allows spins to take on multiple different states, it can be used to simulate multiple biological entities. Each entity can be described by a unique spin state so a lattice can be converted into a representation of a heterogeneous environment

of biological entities.

3.7 Cellular Potts model

The cellular Potts model (CPM) was developed by Glazier and Graner [49] and is a more complex probabilistic CA with Monte-Carlo updating. A cell consists of a domain of lattice sites, thus describing cell volume and shape more realistically. A CPM is a more complex probabilistic CA. Each node is in one of the q states as with the Potts model. Each cell consists of a domain of lattice sites, thus describing cell volume and shape more realistically [1]. The system evolves by updating the cell lattice one node at a time based on a set of probabilistic rules using a spin-based Monte Carlo scheme. A CPM has three primary components:

1. Rules for selecting putative lattice updates.
2. A Hamiltonian or effective energy function that is used for calculating the probability of accepting lattice updates.
3. Additional rules similar to an agent based model.

Rather than just a q-state to describe the system state, each entity can have multiple properties and agent like behaviour making it more suitable for describing biological systems. CPM has been used to study cell differentiation [49], limb development [47] and streaming (see Section 2.7.3) in myxobacteria [165].

3.8 Discussion

Computational resources must be considered when designing a model since these inherently limit its complexity and size. CA models offer speed and the ability to simulate very large numbers of interacting particles/entities; however, discretising events onto a lattice reduces spatial resolution and accuracy. A feature of lattice models is the ability to parallelise execution and evaluate nodes concurrently. CA models lack spatial resolution, which can be introduced at a cost by defining entities spanning multiple nodes and defining complex rules governing how entities move around. CPM

and Monte Carlo models offer a more elegant and descriptive way of describing cell behaviour although it comes with a computational cost. Monte Carlo methods were selected for use in later chapters because they ultimately offer a better way to describe cell physics and their performance still allows thousands of cells to be simulated with relatively modest hardware requirements.

Individual models appear to offer the best features for modelling cell populations (see Section 3.3): a continuous representation of space and discrete representations of space and state making them more computationally efficient to work with. Proceeding chapters will explore how such models can be used to describe the behaviour of myxobacteria.

Chapter 4

The role of phosphate during development

The phosphate assay conducted in this chapter was published as "Phosphate acquisition components of the Myxococcus xanthus pho regulon are regulated by both phosphate availability and development" in the *Journal of Bacteriology* [191].

4.1 Introduction

Myxobacterial behaviour under starvation is affected by the nutrient state of the environment so one of the main areas of research within this project will be to assess the affects of nutrition on myxobacteria colonies.

The developmental phases of myxobacteria requires a form of commitment. Current evidence suggests that once a cell chooses to enters a particular phase (*rippling*, *streaming* or *fruiting*) it commits to being in this phase until it is ready to enter the next phase. It is essential to understand how cells choose to commit to a particular course of action to fully understand the transition from streaming to fruiting. For example, when cells commit to fruiting they do so in the absence of nutrients and continue into the fruiting stage even with the introduction of abundant nutrients.

C-signalling plays an important role in coordinating cell behaviour but its effect occurs later within the development cycle. To begin with, cells are stimulated by more primitive reactions to the levels of particular nutrients in their environment. This ins-

tigates the signalling and the eventual social phenotypes [26].

Phosphate is one of a number of key chemicals required by myxobacteria and they cannot develop without it. Phosphates are available to the bacteria in various forms in the environment, for example in the nucleic acids in DNA; however, there does not appear to be any characterisation of which phosphate sources myxobacteria can utilise for development. Results show that phosphate has an effect on fruiting body formation and so is important in development. The objective of this study was to determine inorganic phosphate sources cells can use for growth. Cells were initially cultured in a double casitone yeast (DCY) rich growth medium and then transferred into media containing phosphate sources. The basis of the phosphate sources was A1W-P, a modified version of the defined medium A1 [22] where potassium phosphate was replaced by potassium chloride to eliminate phosphate (see Section 4.2.2 for details of preparation and composition). This is a synthetic minimal medium which allows the effects of amino acids and carbon and energy sources on development to be distinguishable from each another. A1 contains low levels of isoleucine, leucine and valine which are essential but myxobacteria cannot synthesise themselves. Phosphate sources could then be subsequently added to A1W-P and assayed for their effectiveness as a phosphate source for *M. xanthus*. Figure 4.3 illustrates some preliminary data showing the growth rates of DK1622 cells in the growth and minimal media in order to test that the strain grew correctly.

An assay to look at how the bacteria responds to various phosphate sources was undertaken looking at how phosphate affects *fruiting body* development, *motility* and growth.

4.2 Materials and methods

The basis of all media used to test phosphate compounds was A1 [22, 97]. Phosphate compounds were added A1 to ensure that phosphoryl groups were present at a concentration of 1 mM.

All experiments used *M. xanthus* strain DK1622. Cells were initially incubated at

30 °C in liquid DCY[1] growth medium until they reached an optical density (OD) of 1.0 at 600 nm ($OD_{600\,nm}$). An $OD_{600\,nm}$ of 1.0 indicates cells are in their exponential growth phase meaning cells are growing and dividing normally and can survive being transferred to other growth media.

Optical density (OD) gives a measure of the cell density and is a measure of the transmittance of an optical medium for a given wavelength. It is determined from the amount of light incident on a sample and amount of emitted light from the sample. Optical density is defined as:

$$OD_\lambda = log_{10}\left(\frac{I_o}{I}\right) \tag{4.1}$$

where I_o is the intensity of light of wavelength λ incident on the sample and I is the intensity of radiant energy transmitted by the sample. Cells normally require a minimum of 72 h incubation to achieve an $OD_{600\,nm}$ level of 1.0.

All experiments were conducted in triplicate. Cells were centrifuged to form a compact pellet and then re-suspended in liquid A1W[2] media without phosphate to wash off any phosphate that was present from the DCY so that all cells start each assay in a starvation media. Cells were centrifuged once more and then transferred into 10 ml of each growth media. Each sample was incubated at 30 °C and the OD level of each sample was measured once every 24 h to assess how well cells were growing in each medium.

4.2.1 Growth media

M. xanthus cells were cultured in both DCY growth media and TPM starvation media at 30 °C for comparison. The composition of DCY and TPM is given in Table 4.1.

4.2.2 Phosphate growth media

To test the effectiveness of phosphate sources, two defined media A1W and A1W-P were used (see Table 4.2). Both were adapted from the defined A1 [22, 138]. The media

[1] Double casitone yeast extract medium. A rich growth medium for the initial growth of cells in large numbers.

[2] A minimal growth media containing only essential nutrients.

Table 4.1: DCY and TPM growth media for initial cell cultures.

	Ingredient	Concentration
DCY	Casitone	20 g/l
	Yeast	2 g/l
	$MgSO_4$	8 mM
	Tris-HCl pH 7.6	10 mM
TPM	Tris	10 mM
	KH_2PO_4	1 mM
	$MgSO_4$	8 mM

are derived from initial studies by Dworkin [36] looking at the nutritional requirements of *M. xanthus*.

Table 4.2: A1W and A1W-P minimal growth media. A plus symbol (+) indicates the presence of an ingredient, a minus symbol (-) indicates it was omitted from the solution.

Ingredient	Presence		Concentration
	A1W	A1W-P	
L-asparagine	+	+	100 μg/ml
L-isoleucine	+	+	100 μg/ml
L-leucine	+	+	100 μg/ml
L-Methionine	+	+	100 μg/ml
L-Valine	+	+	100 μg/ml
L-Phenylalanine	+	+	100 μg/ml
Sodium Pyruvate	+	+	5 g/l
Aspartic acid	+	+	100 μg/ml
Tris	+	+	10 mM
KCl	-	+	10 μg/ml
KH_2PO_4-K_2HPO_4	+	-	1 mM
$MgSO_4$	+	+	8 mM
$FeCl_2$	+	+	10 μM
$CaCl_2$	+	+	10 μM
$(NH_4)_2SO_4$	+	+	0.5 mg/ml

In addition to A1W and A1W-P, two other defined media, M1 and M1-P [193], were also used to test the effectiveness of phosphate sources for growth. These media were designed to act as controls to verify that phosphate sources were used for growth and in case A1W and A1W-P proved to be inadequate growth media. The composition of M1 and M1-P is listed in Table 4.3.

Table 4.4 lists the phosphate sources that were tested in A1W-P.

Table 4.3: M1 and M1-P growth media. A plus symbol (+) indicates the presence of an ingredient, a minus symbol (-) indicates it was omitted from the solution.

Ingredient	**Presence**		**Concentration**
	M1	**M1-P**	
$MgSO_4$	+	+	0.4 μM
$NaCl_2$	+	+	200 μg/ml
$FeCl_3$	+	+	7.4 μM
$CaCl_2$	+	+	18 μM
Alanine	+	+	1000 μg/ml
Arginine	+	+	100 μg/ml
Asparagine	+	+	500 μg/ml
Cystine	+	+	100 μg/ml
Glycine	+	+	100 μg/ml
Histidine	+	+	100 μg/ml
Isoleucine	+	+	1000 μg/ml
Leucine	+	+	2000 μg/ml
Lysine	+	+	500 μg/ml
Methionine	+	+	500 μg/ml
Phenylalanine	+	+	1000 μg/ml
Proline	+	+	100 μg/ml
Serine	+	+	200 μg/ml
Threonine	+	+	100 μg/ml
Trypophan	+	+	400 μg/ml
Tyrosine	+	+	400 μg/ml
Valine	+	+	200 μg/ml
Tris	+	+	10 mM
KH_2PO_4	+	-	1 mM
KCl	-	+	1 mM

Table 4.4: Phosphate sources tested in A1W-P medium

	Phosphate source
	glucose-6-phosphate
	glyceraldehyde-3-phosphate
	glycerol-3-phosphate
	phosphatidylethanolamine
	phosphatidylcholine
	phytic acid
	sodium polyphosphate
	sodium pyrophosphate
Nucleotides	ATP
	Sigma-Aldrich® Calf Thymus DNA
	deoxynucleoside triphosphate
	DNTP
	nucleoside triphosphate
	RNA
Phosphonates	methylphosphoric acid
	phosphonomethyl glycine
	3-N-phosphomethyl glycine
	phosphonoacetic acid
	2-amino-3-phosphonopropionic acid
	2-aminoethylphosphonic acid

4.2.3 Stock solutions

All stock solutions were made at 1000x concentration, with stock solutions added to media in the appropriate quantities for 1x concentration. Phosphate solutions used during experiments were prepared in 100 ml batches to prevent contamination. Solutions were prepared from stocks and pure water, sealed and autoclaved for 15 min at 121 °C. Each 100 ml batch of solution was used to prepare three replicates for each experiment. DNA and DNTP solutions were filter sterilised as heat treatments would have caused denaturing and hydrolysis.

4.2.4 *M. xanthus* DK1622 resurrection

Wild type cultures were resurrected from freezer stocks using the protocol outlined in Figure 4.1. Cells were grown on DCY agar in batches of ten plates for two days in an

oven at 30 °C before being transferred to a fridge at 4 °C for storage. Each plate was sealed before storage to prevent contamination. New plates were prepared every three weeks to replace older cultures. Cell cultures for all experiments were obtained from these plates.

Protocol for *M. xanthus* DK1622 resurrection

(1) Prepare an agar plate made from DCY and agar. Agar should be at a concentration of 15gdm^{-3}.
(2) Remove cells from -80°C freezer.
(3) Immerse inoculation loop in ethanol.
(4) Use a gas burner to burn off excess ethanol and sterilise loop.
(5) Scrape the clean loop in the cell culture to deposit a small volume of cells onto it.
(6) Use the loop to lightly spread an even trail of cells over the DCY agar plate.

Figure 4.1: Protocol for *M. xanthus* resurrection from freezer stocks.

4.2.5 Culture preparation

M. xanthus DCY cultures were centrifuged in a Wifug™ Labor 50M table top centrifuge for a minimum of ten minutes to allow a compact pellet of cells to form at the bottom of the tube. Once the pellet had formed, 9 ml of the supernatant was poured off and the pellet was washed in 9 ml of AlW-P to remove traces of external phosphate from the cells. Cells were then compacted and washed twice more to leave a 10x concentration of cells suspended in 1 ml of A1W-P. 100 μl of concentrated cells was added to each phosphate solution to give them a starting optical density (OD) of 0.1 at 600 nm ($OD_{600\,nm}$).

4.2.6 Measurement of cell growth

Cultures were grown in 30 ml plastic Universal tubes. OD readings were taken at $OD_{600\,nm}$ using a Ultrospec™ 3000 pro UV/Visible Spectrometer. Figure 4.2 details

the standard protocol for taking measurements that was used [8, 132]. Readings were taken in one ml plastic cuvettes. Cultures were pipetted from the Universal tubes into cuvettes. Reference values for calibration were obtained by measuring the OD of each phosphate solution without a cell culture in it.

Protocol for taking spectroscopy readings

(1) Switch on tungsten bulb spectrophotometer.
(2) Allow 15 minutes to warm up before taking readings.
(3) Set wavelength to 600 nm.
(4) Zero the meter to calibrate it.
(5) Fill a cuvette with 1 ml A1-P media and take OD reading as a reference.
(6) For each sample, fill a cuvette with 1 ml of sample and take OD reading.
(7) If the OD is greater than 1.0 check for contamination.

Figure 4.2: Protocol for taking optical density measurements.

4.3 Results

Figure 4.3 shows that cells grew best in the DCY media (the non linear response of the log graph indicates rapid growth). Cells also grew in the A1W media which was also designed to allow growth. Growth in A1W-P (A1W without phosphate) was significantly less than growth in A1W indicating that phosphate is a key nutrient. The better a curve matches the profile for A1W, the more readily *M. xanthus* can use the phosphate source to grow.

Having established that *M. xanthus* uses phosphate and its effect is quantifiable, the next step was to identify substances that could act as potential phosphate sources. The substances tested are given in Table 4.4. Compounds were chosen based on their likelihood of existing in an accessible form in a myxobacterial habitat either because of lysis of prey organisms or because of their natural presence in the soil environment. Figure 4.4 shows a time course of the growth of *M. xanthus* in starvation medium A1W-P (without phosphate) to which a number of different phosphate sources were added.

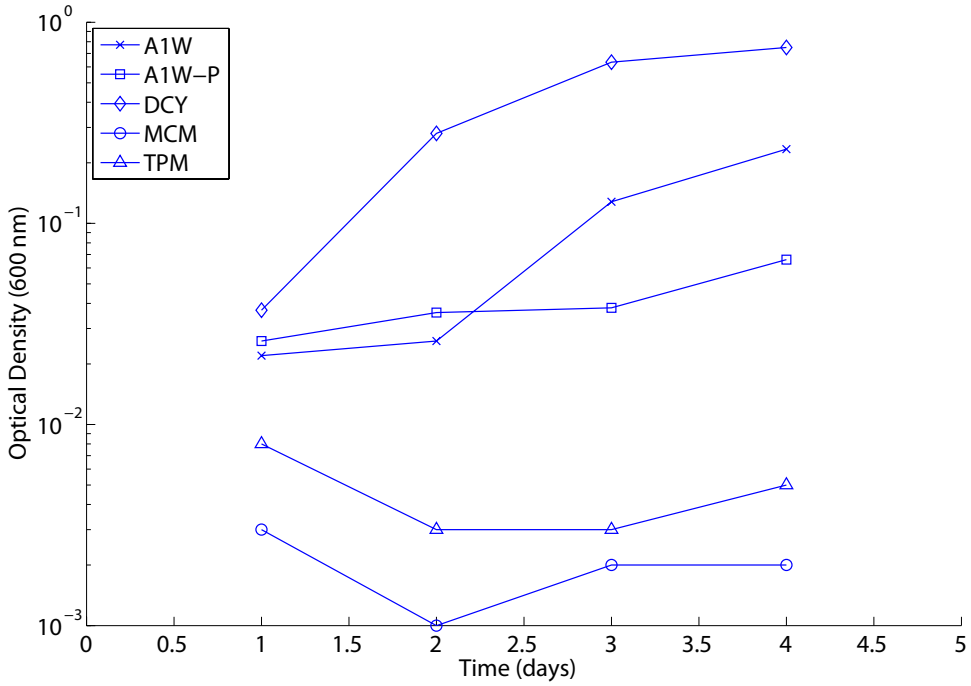


Figure 4.3: The effects of phosphate sources on the growth of *M. xanthus*.

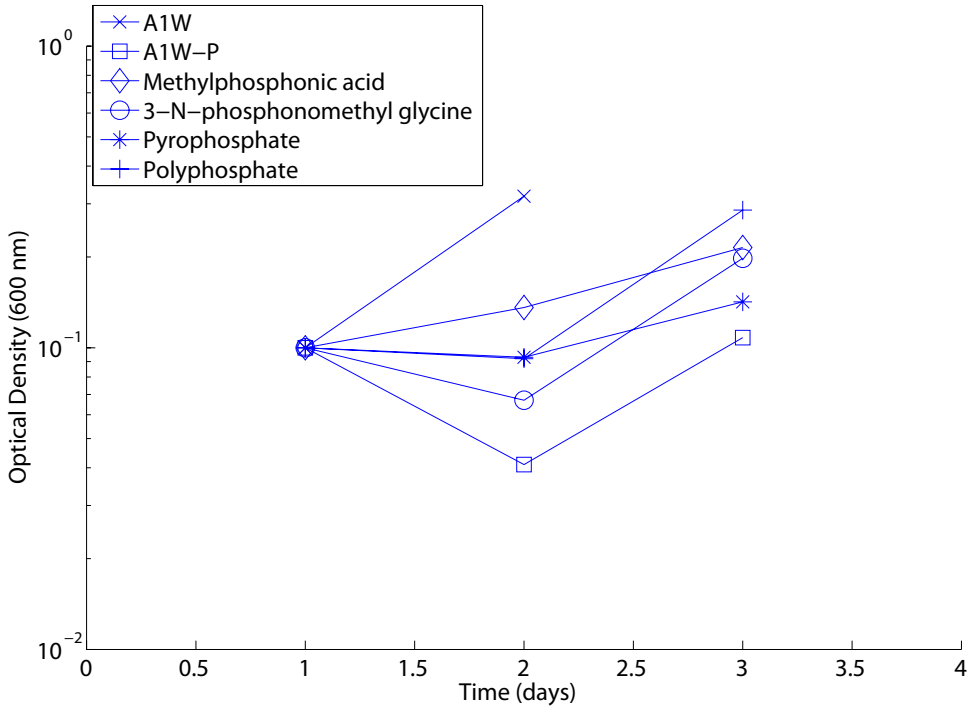


Figure 4.4: Phosphate usage experiment set one. OD values were normalised to 0.1 at day one, and the values are the averages of three independent experiments. For clarity error bars are not shown.

There was growth in A1 and no growth in A1W-P. There was significant growth in both methylphosphonic acid and pyrophosphate. The profile of their growth curves better matched that of A1W (which is a growth media) indicating the bacteria could utilise these phosphate sources.

A substance was classified as a potential phosphate source for *M. xanthus* if its inclusion in A1W-P allowed the $OD_{600\,nm}$ (or number of CFU[3]) of a culture to at least double after seven days of incubation. When this criterion was used, only pyrophosphate, polyphosphate, glycerol-3-phosphate, and glyceraldehyde-3-phosphate were classified as potential phosphate sources for *M. xanthus*. Figure 4.8 shows the log of the normalised optical density versus time for the identified sources to make any non-linear relationships more apparent. For the growth sources mentioned they display a non-linear trend indicating significant growth.

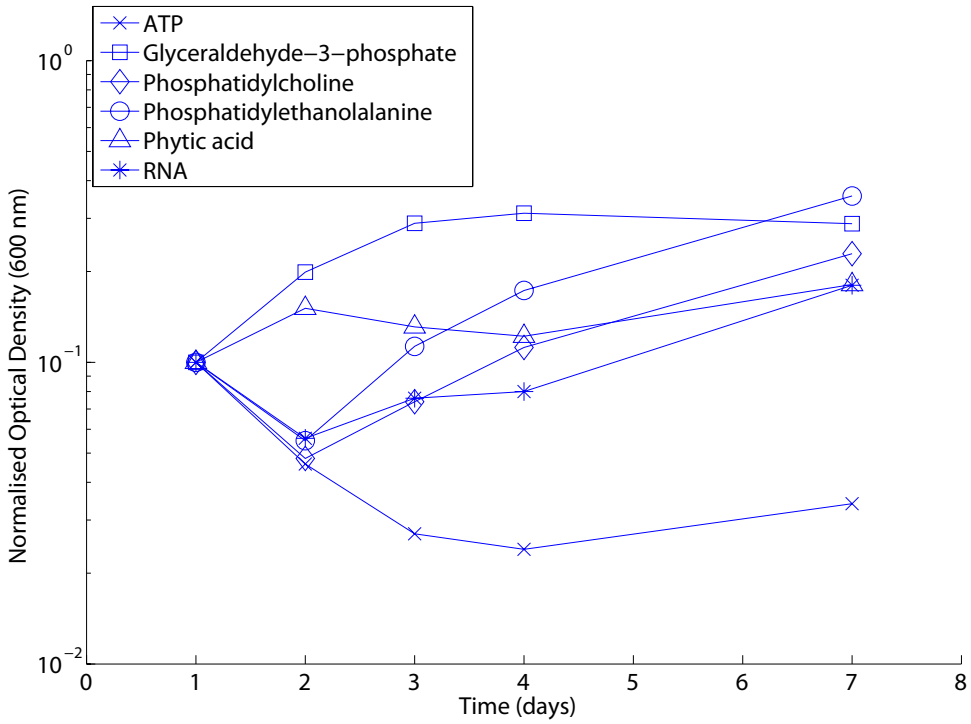


Figure 4.5: Phosphate usage experiment set two. OD values were normalised to 0.1 at day one, and the values are the averages of three independent experiments. For clarity error bars are not shown.

Figure 4.4 shows a time course of the growth of *M. xanthus* in starvation medium

[3]Colony-forming unit is a measure of viable cells in which colonies, representing aggregates of cells, are counted, rather than individual cells.

A1W-P (without phosphate) to which a number of different phosphate sources were added. There was growth in A1W and no growth in A1W-P. There was significant growth in both methylphosphonic acid and pyrophosphate and polyphosphate. The profile of their growth curves better matched that of A1 (which is a growth media) indicating the bacteria can utilise these phosphate sources.

From experiment set two (see Figure 4.5) only glyceraldehyde-3-phosphate appears to cause cell growth with clear growth from the start. The downward trend of the OD from days four to seven indicates the sample was not contaminated so growth can be attributed to glyceraldehyde-3-phosphate. The other phosphate sources were not considered to induce growth because of the marked decrease in the OD at the start, suggesting that any subsequent growth was not caused by the phosphate sources.

From experiment set three (see Figure 4.6) only glyceraldehyde-3-phosphate appeared to cause cell growth, which agrees with experiment set 2. The average OD of the samples after seven days was 0.726 indicating growth was not due to contamination.

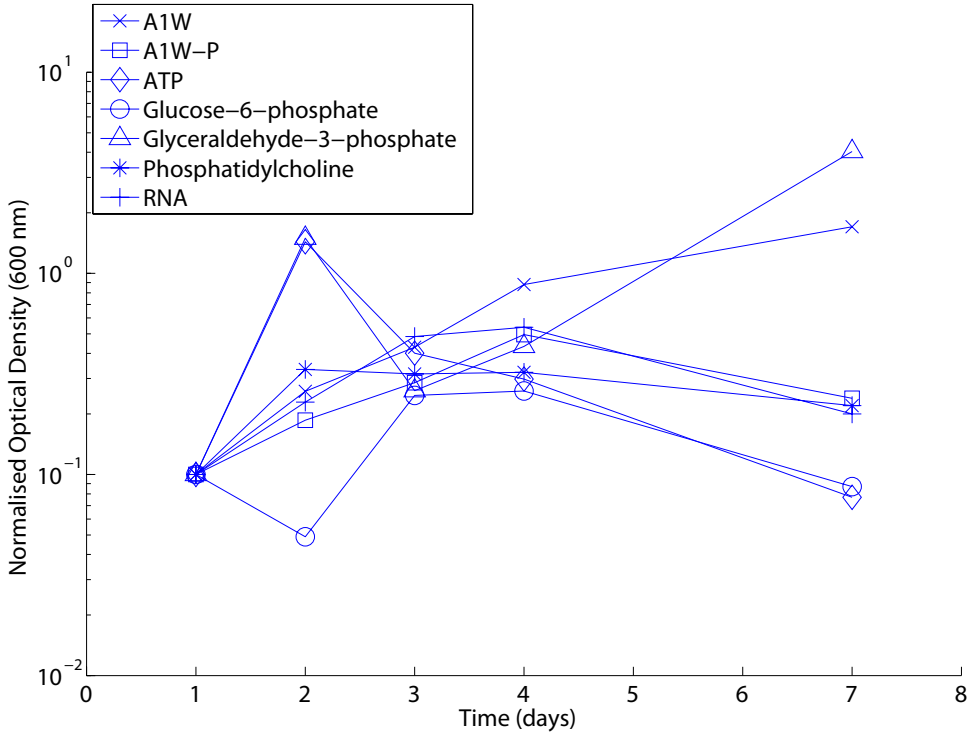


Figure 4.6: Phosphate usage experiment set three. Optical density values were normalised to 0.1 at day one, and the values are the averages of three independent experiments. For clarity error bars are not shown.

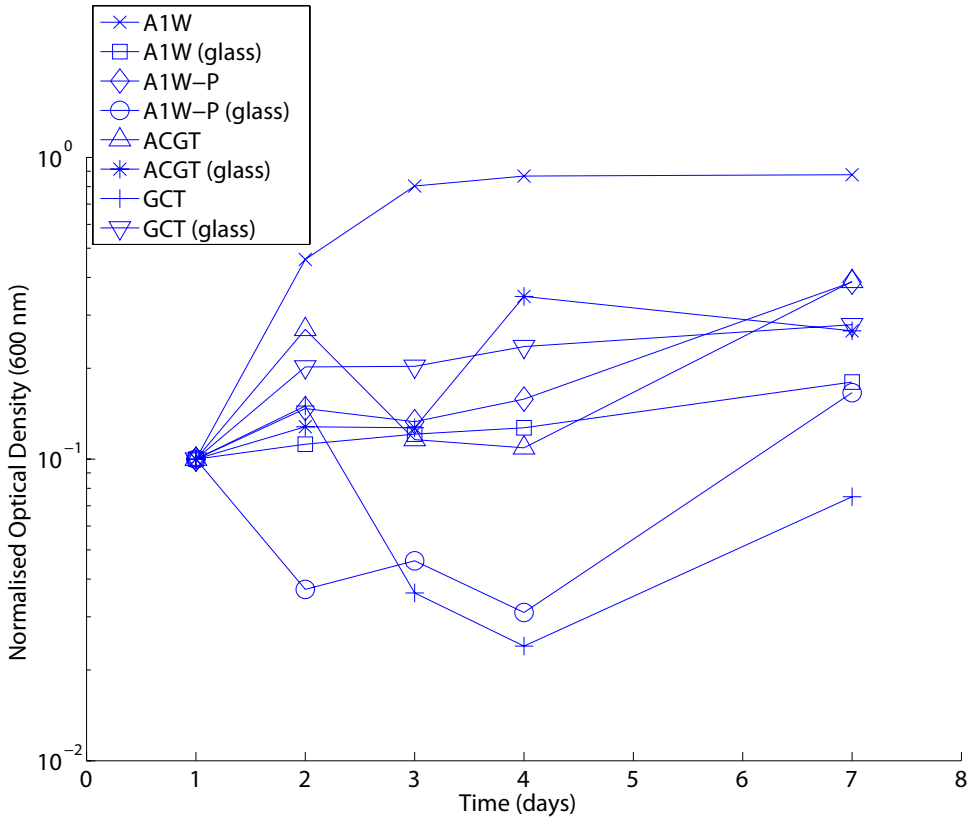


Figure 4.7: Phosphate usage experiment set four. Optical density values were normalised to 0.1 at day one, and the values are the averages of three independent experiments. Cells were tested with phosphate compounds in plastic and glass tubes (as indicated) to assay for differences in growth. For clarity error bars are not shown.

4.4 Discussion

Although some of the phosphate sources tested were previously assayed for their effects on myxobacteria [114, 203], these experiments provide a more comprehensive, quantifiable measure of which substances specifically promote growth.

Inorganic polyphosphate may be a likely candidate for promoting growth due to *M. xanthus* being dependent on five signalling genes: *asg*, *bsg*, *csg*, *dsg*, and *esg*. These genes control development and are used by the cell to sense starvation and begin fruiting and sporulation. The protein encoded by *bsg* is a partial homolog for Lon, a protease in *E. coli* [203]. Polyphosphate binds to Lon and increases its activity 20-fold which generates essential amino acids for protein synthesis. *E. coli* recruits phosphate when it is introduced into a starvation media to survive. If *bsg* is behaving in a simi-

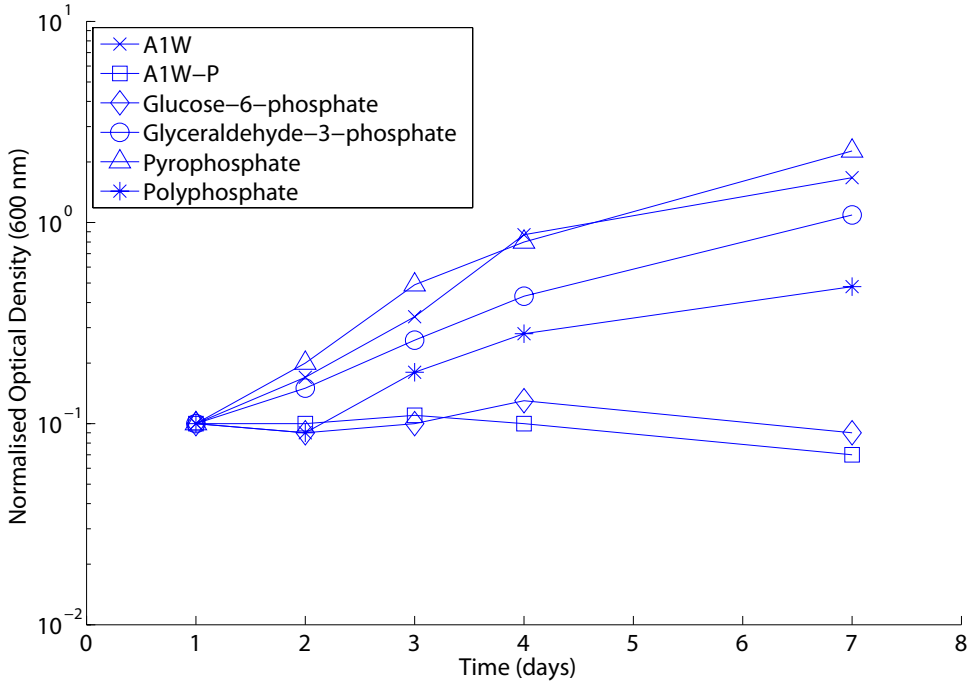


Figure 4.8: Phosphate sources identified as causing growth in *M. xanthus*. Changes in cell number were monitored by recording the $OD_{600\,nm}$ at different times after inoculation (at zero time) into minimal medium (A1W-P) with different phosphate sources. Optical density values were normalised to 0.1 at day 1, and the values are the averages of three independent experiments. For clarity error bars are not shown; however, the standard deviation was within 15 % of the sample mean in all cases.

lar way, introducing polyphosphate to starved *M. xanthus* would result in the growth observed.

Watson and Dworkin [185] showed that glyceraldehyde-3-phosphate is present in fruiting bodies and is highly active, suggesting it is important to the life-cycle indicating it could be a potential growth source. Myxobacteria cells contain glycerol-3-phosphate acyltransferase (PlsB) homologues [30] which produce phosphatidylethanolamine to support growth. This again strongly indicates that glycerol-3-phosphate is a likely growth source. It is interesting to note that whilst glycerol-3-phosphate appears to promote cell growth, it has been found to inhibit swarming and aggregation [114]. Those experiments were conducted on plate media where growth was not being assayed whereas the experiments conducted for this work were in liquid media so the affects on swarming and aggregation were not assayed. However, the experiments support the idea of glyceraldehyde-3-phosphate being a growth source. Growth in po-

lyphosphate is lower than in glyceraldehyde-3-phosphate. This may reflect that *bsg* only shares a 45 % sequence similarity to *lon* and may therefore not be as efficient at cleaving proteins for use [46]. The response may also be limited to prevent excessive growth in an environment that cannot support it.

Myxobacteria lyse prey cells for nutrients, so calf thymus DNA and DNTPs were also tested to see whether *M. xanthus* could obtain phosphate directly from nucleotides (see Figure 4.7). Interestingly there was no significant change in results so it is probable cells do not obtain phosphate directly from their lysed products. Cells were grown in both plastic and glass 10 ml universal tubes to assay whether plastic was affecting cell growth but there was no significant, reliable difference between samples.

Although myxobacteria grows in DCY, one of its constituent components, casitone (hydrolysed casein), is capable of inducing a phenomenon called *cooperative growth* where cells exhibit increasing growth rates as a function of increasing cell numbers [78, 140]. Cooperative growth was triggered in dense populations when the medium contained hydrolysed casein at a concentration of 0.2 % by mass. As casitone is present in DCY at 2 % by mass, cells were washed twice in A1W-P to ensure the removal of DCY to stop it interfering with the phosphate usage. It was not essential to remove all the casitone since below approximately 0.01 % by mass, there is not enough to support growth so any observed growth would be due to other factors.

The experiments were initially performed on wild type strain DK1622 cells. DK1622 is a model strain of *M. xanthus* that is commonly used for laboratory experiments. The strain has undergone many transformations and the deletion of a 250kb region of its genome. To test whether the results were specific to this particular strain, experiments were also performed on a derivative strain of DK101 which carries two different plasmids: DK101 [pADH206] and DK101 [pMAR217].

One caveat of these experiments is that they were carried out in laboratory conditions and this may, in itself, have an effect on cell behaviour. Although cells glide across an agar surface for example, they typically will not be able to do so in a soil environment or least not with the same efficacy. Similarly, although cells were grown in liquid media, this is another atypical environment for cells. There is the possibility that im-

mersion in liquid and factors such as stress responses to rapid changes in nutrition and environmental conditions affects cell behaviour and what is observed does not happen to the same extent in a natural habitat. This discrepancy between laboratory and environmental conditions is a general problem affecting experimental biology; however, further negative control experiments could be devised to assess these problems.

If a population of *M. xanthus* cells is transferred to medium lacking phosphate, growth ceases within one doubling time (24 h in A1W). This suggests that the amounts of phosphate stored in the cells (usually polyphosphate) are small. This agrees with the results of polyphosphate assays, which show that polyphosphate levels are low in exponentially growing cells, although polyphosphate transiently accumulates during early development [203].

Cessation of growth required the concentration of phosphate to be reduced below 10 mM and was unaffected by previous incubation in different media, whether the medium was rich (DCY) or minimal (A1W) [191].

M. xanthus is able to grow with other microorganisms as sole nutrient sources [127]. Therefore a number of phosphate-containing biomolecules that might be expected to be released by lysis of prey were tested to determine their abilities to act as sole phosphate sources. Surprisingly, nucleic acids, phospholipids, and nucleotides could not act as sole phosphate sources in the assays, nor could glucose-6-phosphate. Phytic acid and phosphonates were also tested, as phytase and phosphonate transporter genes have been annotated in the *M. xanthus* genome.

The assays showed that pyrophosphate, polyphosphate, glycerol-3-phosphate, and glyceraldehyde-3-phosphate could be utilized by *M. xanthus* as sole phosphate sources. With polyphosphate, there was a lag phase prior to exponential growth longer than that observed with phosphate itself (Figure 4.8). This suggests that there is a delay in utilization, which may be a consequence of the requirement for accumulation of secreted polyphosphatases prior to uptake of liberated phosphate.

The ability to utilize glycerol-3-phosphate and glyceraldehyde-3-phosphate as sole phosphate sources suggests that there is uptake of glycerol-3-phosphate by cells, which can be readily produced from glyceraldehyde-3-phosphate, presumably liberated by

extracellular lipolysis of prey organisms.

The doubling time in the A1W cultures was 24 h; however, in a rich medium *M. xanthus* doubles every 3 h, so the results presented here cannot be directly extrapolated between minimal and rich media. Certainly other factors are affecting growth in a rich medium and the relationship between nutrients and growth becomes more complex. The phosphate sources tested are most likely to come from prey cells in a natural environment so it would be interesting to study phosphate usage during predation. This would require a different experimental protocol since growth in liquid media would be contaminated by prey and it would be difficult to quantify either *M. xanthus* or the prey bacteria. It is also likely that cells can use multiple phosphate sources simultaneously so the effects of different biomolecules in combination could be assayed.

Chapter 5

The design of an agent based simulation framework

The data collected in Chapter 4 provided evidence of the most likely phosphate sources myxobacteria will use in a vegetative environment. This information in itself does not explain how the observed social behaviours manifest but it can form the basis of models which address cell morphogenesis.

Theoretical modelling of biological systems can elucidate properties of a system that may be otherwise difficult to investigate using purely biological analysis. As a simple example, consider tracking the behaviour of individual cells within a population. Fluorescence tagging of cells allows us to track up to a few hundred cells, but this may represent a very small percentage of the whole population. Tagging also limits resolution since the microscope must keep a relatively large tracking area in focus so that all of the cells can be captured. To achieve higher resolution, electron microscopes can be employed but they cannot monitor a system in real time. These techniques also require the use of chemicals and probes which may disrupt the system being studied. A theoretical modelling approach can allow whole populations to be simulated and data collected on individual cells. It also makes it easier to perform experiments under different conditions to get consistent feedback from the model.

Chapter 3 outlined the role of theoretical modelling and the common methodologies employed to design and analyse systems. Although concepts such as CA, LGCA,

CPM and Monte Carlo dynamics were introduced, there is a *gulf of execution*[1] between theory and actual implementation. Developing a computational model using these techniques and subsequently applying it to study bacteria is non-trivial. This chapter describes a concrete implementation of a software modelling framework employing the methodologies outlined previously, which can be used to answer biological questions. Although initially developed to specifically study myxobacteria, it has been generalised for studying a wider class of problems.

5.1 Introduction

The behaviour of bacterial populations is a result of the heterogeneous behaviour of the internal signalling pathways of each cell. Even within a homogeneous population, individual cells can exhibit different behaviour to their peers. Multicellular development in organisms requires the coordination of cell movement and it would be desirable to be able to explore how system parameters affect the cell population. It can be non-trivial to assay such systems *in vivo* and study how heterogeneous cell populations interact. One approach to studying cell behaviour is to use computation models of cells since the cost of creating virtual cells is constant and it is relatively straightforward to have a population of cells in different states.

Numerous computation packages exist to model biological behaviour which model at one of three main scales of interest: molecular, cellular or inter-cellular. Typically a package will either focus on modelling the biology of individual cells [170] or else look at the behaviour of cell populations [69] using simple representations of individual cells so that large numbers can be simulated efficiently.

Biological systems are dynamic environments in which are in a constant state of flux, for example protein levels or cell birth and death. A differential equation based approach (see Section 3.1) to represent dynamic structural changes generally requires convoluted mathematical techniques. Object-oriented (OO) representations of biological entities make it much easier to explore the internal and external states of entities and

[1]Gulf of execution is a term commonly used in human computer interaction to describe the discrepancy between theory and practice; having a desired goal does not necessarily imply the goal is easily achievable.

monitor dynamic properties [172]. OO paradigms encapsulate entities as individual objects, each with its own set of properties and behaviours. This offers an inherently intuitive way of describing a system in a way that resembles how we naturally reason about how real systems behave. A myxobacteria cell for example can be thought of as a self contained entity with properties such as length, flexibility and location of motility systems. There is a direct mapping between the cell and a virtual, object representation of the cell.

In this chapter a framework for multi-agent based cell simulations (FABCell) [59] is presented. FABCell is a general purpose computer modelling environment for simulating cell dynamics and population behaviour. It is a highly modular object-oriented (OO) framework which allows the creation of user-defined agent models. FABCell was written in C++ because of its speed of execution [145] and portability allowing the code to be compiled and run on a variety of hardware platforms. The framework was created to run different types of model within the same environment, using a consistent interface, to investigate aspects of regulated motility in myxobacteria. FABCell is designed to consolidate and run different types of model to make it easier and faster to explore modelling ideas. Having a unified way of creating models reduces the learning curve for users since they only have one methodology to learn and it is easier for them to choose the modelling technique most appropriate for a problem.

FABCell is agent-focused, with each cell acting as an independent entity. It uses a modular environment and interaction system to simulate agents and can be used to run Cellular Automata (CA), Lattice Gas Cellular Automata (LGCA), Cellular Potts Model (CPM) and off-lattice simulations.

5.2 Requirements analysis

FABCell was conceived as a software tool rather than a bespoke program to solve a specific task. In order to realise its design, a rigorous software engineering approach [158, 178] was adopted during the design and development.

The process of designing software begins with the requirements analysis to establish the functionality it must possess [179]. The *user requirements* phase considers

each type of user who might interact or use a system since this determines its functionality. By outlining what the software must do, a list of goals can be produced that the software can be benchmarked against to ensure it performs as designed. Note that requirements analysis does not mandate technical implementation details; it merely specifies the general goals and functions of the system at an abstract level. It is common practice to use the *Unified Modelling Language* (UML) during design to provide a visual, diagrammatic representation of a system. The notation of Priestley [129] has been adopted in this section as a reference standard to ensure consistency in the structure and nomenclature of UML diagrams.

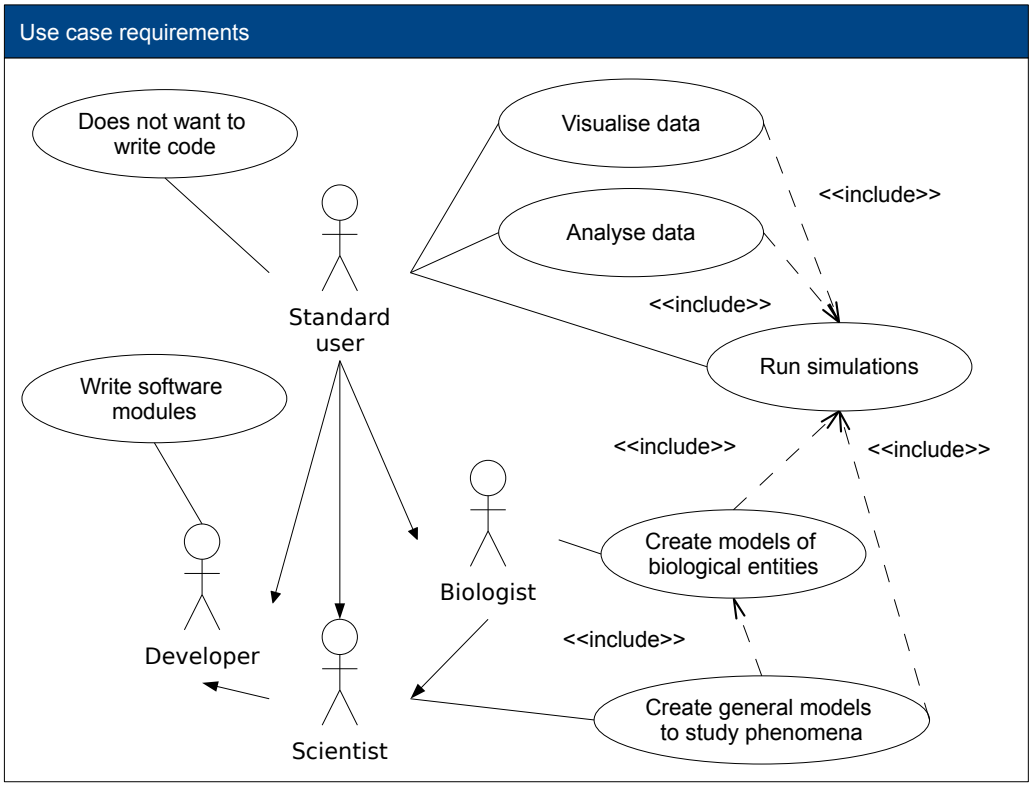


Figure 5.1: FABCell use case requirements indicating the primary functions of the software and the users it was designed to serve. There are two broad user groups: technical users who can program and work with FABCell directly and non-technical users who want a simpler user interface. These two groups can be subdivided into niche subgroups with specific requirements.

Figure 5.1 presents a UML *use case*[2] scenario for the major user types (or actors in UML parlance) who might use FABCell. The *standard* user represents how an average

[2]A use case is a description of a systems behaviour as it responds to a request that originates from outside of that system. It describes *who* can do *what* with the system in question.

user might use FABCell; namely to create a simulation and run it, visualise data produced by the simulation and perform analysis on the data. Crucially, standard users are not likely to be interested in the design and implementation of the system, only how to use it; they do not want to concern themselves with programming. A sub group of the *standard* users are the *scientists* who want to use FABCell as a scientific tool. FABCell is targeted towards the research community. A special consideration is given to *biologists*, a specialisation of the *scientist* role. *Biologists* are mostly likely to want to create agent based models to replicate biological phenomena. This requires more complex functionality than just running a conventional cellular automata for example. The most complex user is the *developer* who inherits all the needs of the other users but additionally might want to develop and write their own code to work in FABCell. Developers are an important consideration especially in an open source, community tool as they can add missing functionality and are more likely to be early adopters of the system who can provide feedback and disseminate information about FABCell to other users.

5.3 Existing simulation tools

There are already a number of existing software modelling packages available so this section addresses why FABCell was created. A review of agent modelling tools listed by the Swarm Development Group [51] suggests that the majority of modelling packages available are Java™ based [107]. Some of the more well known tools are described in Table 5.1.

There is a long running debate within the computer science community over the performance of Java™. Early versions of Java™ ran code far slower than equivalent C/C++ implementations and suffered from large memory overheads from running the Java™ Virtual Machine[3] (JVM). More recent versions of Java™ are faster; however, they still do not match the performance of C/C++ [166]. Whether languages such as Java™ and C# are significantly slower than C/C++ in real world applications will

[3] A virtual machine, such as the JVM, is a software implementation of a machine that executes programs like a real machine allowing programs to maintain an abstraction from the actual hardware they are running on. This allows software to run on multiple hardware platforms without having to be compiled specifically for each.

Table 5.1: Existing biological simulation tools.

Name	Language	Notes
AgentCell	Java™	[38]
Uses Repast to inherits any flaws present in that toolkit. Very limited documentation to explain how to adapt the system.		
Compucell	C++	[69]
A C++ framework for running CPM models. It is limited to running lattice based CPM models and is not well documented, making modifications difficult.		
E-Cell	C++	[176]
Fast and efficient since it is written in C++. Supports modelling and simulation of biochemical and genetic processes to study the functions of proteins, protein-protein interactions, protein-DNA interactions, regulation of gene expression and other features of cellular metabolism. Does not support spatial simulations of cells.		
Java™ Agent Development Framework (JADE)	Java™	[68]
Focusses on peer-to-peer networked agents for applications and conventional agent applications. Would require significant work to make it more high performance and suitable for simulating biological entities.		
MASON	Java™	[96]
Uses Java3D™ for visualisation limiting the number of agents that can be simulated. Tool for programmers so would require extensive modification to make it more user friendly.		
Repast	Java™	[91]
Tool for programmers so would require extensive modification to make it more user friendly.		

probably remain open to debate for some time; however, in high performance computing environments, Java™ is rarely (if ever) used for computationally intensive tasks. Cluster hardware is traditionally optimised for more scientific languages meaning that large scale distributed models would be better implemented in supported languages such as C/C++.

A deficiency of all the modelling tools investigated was their general purpose nature. They appear to have been designed ostensibly for a theoretical agent modelling community. The needs of a biological simulation tool are slightly more esoteric and specialised, and in all cases it would have required the extension and further development of all of the tools in order to realise the models in Chapters 6 to 8. The amount of work required to repurpose any one of the tool kits is unlikely to have been significantly less than the development of a tool specifically for dealing with biological problems.

At a more technical level, many of the tools use threading so that agents can run concurrently with each agent running as a thread. This is acceptable for a small number of agents, but in a simulation of thousands of agents, the overhead in constantly switching between threads means that the benefit of parallel execution is lost and it may not even be possible to have that many agents running. Further work would be required to get a more refined control of how the tools use concurrency.

5.4 System design

One of FABCell's design goals was to link internal cell behaviour with external macroscopic behaviour. Tools such as E-cell[4] [173] exist to model internal parameters using reaction and differential equations but they do not cope with the spatial distribution of cell populations; however, their work flow models make a suitable starting point for designing the internal behaviour mechanisms of cells in FABCell.

[4]See also `http://www.e-cell.org/`

5.4.1 Architecture

FABCell comprises over 500 classes organised into a hierarchy of libraries (see Figure 5.2). The libraries in one level build upon the functionality of those in the lower levels. For brevity, only principle components will be described in detail. The core classes of the major libraries and their relationships to each other are detailed in Figure 5.3.

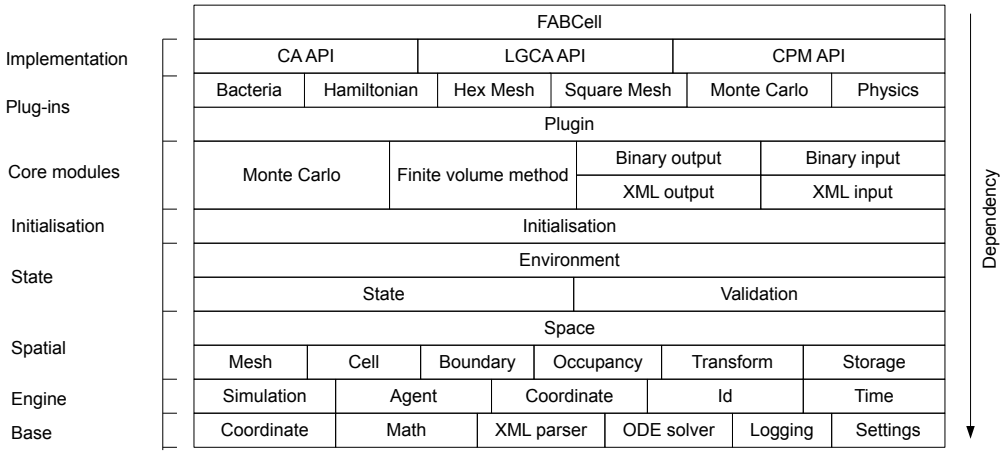


Figure 5.2: FABCell library structure. FABCell is designed as a set of libraries that are organised in a hierarchy. The Tools and Base libraries provide commonly used functions which the other libraries build upon to create the lattice and agent framework.

The core classes provide a skeletal implementation that technically allows simulations to be run; however, users are required to implement a lot of additional components to get a usable simulation working. A large number of the FABCell libraries are designed to provide extra functionality to make FABCell user-friendly. They are technically libraries since they are non-critical, but are, in fact, utilised in most simulations. Figure 5.4 details the classes in core peripheral components. These offer functionality, such as data input/output (I/O) and a plug-in framework for extending FABCell's functionality.

5.4.1.1 Core FABCell components

A FABCell simulation comprises multiple interacting objects, which are outlined in this section. An overview of what the components do is given rather than a detailed

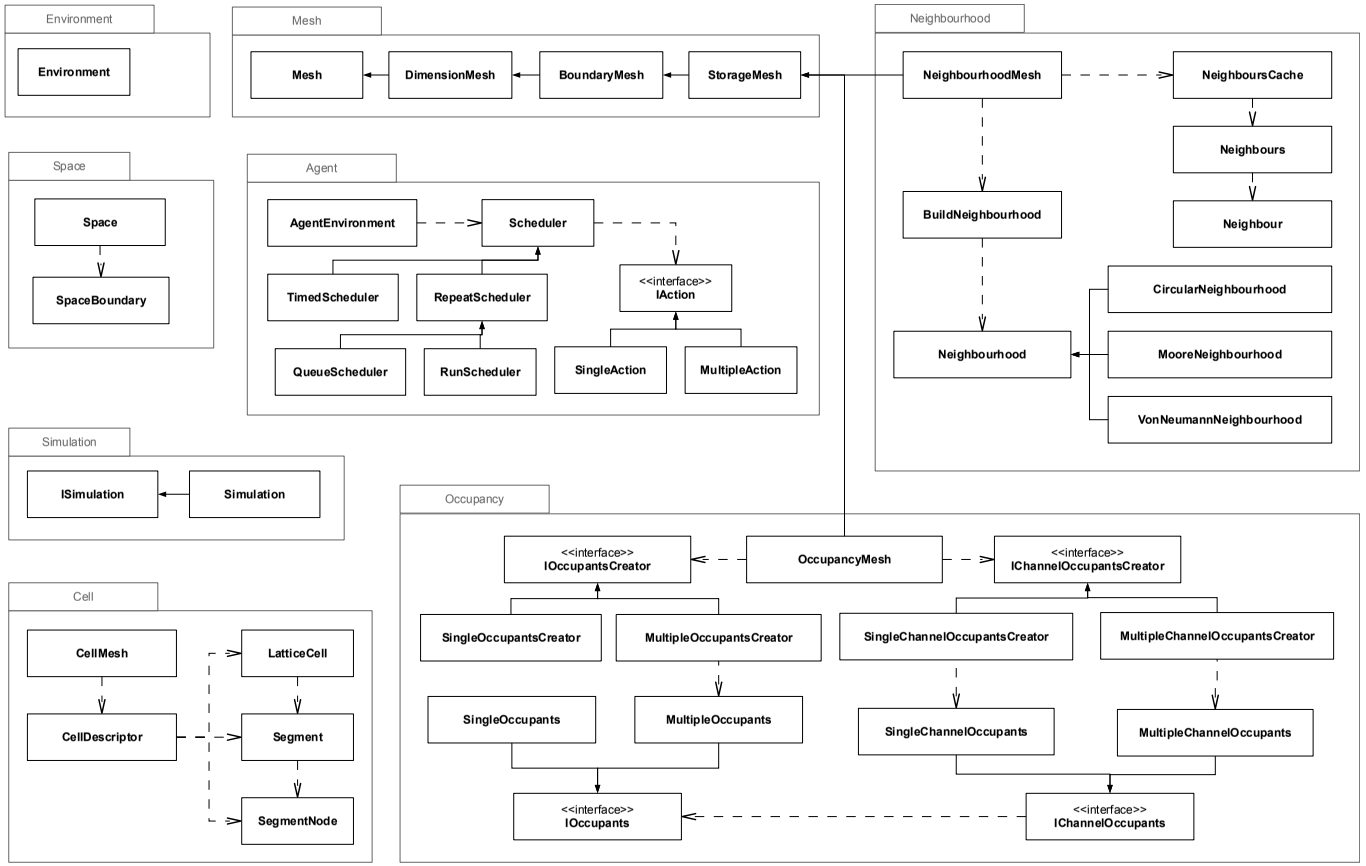


Figure 5.3: Class diagram of the core components of FABCell. FABCell comprises over 500 components so this diagram represents the most important classes which are used in every simulation.

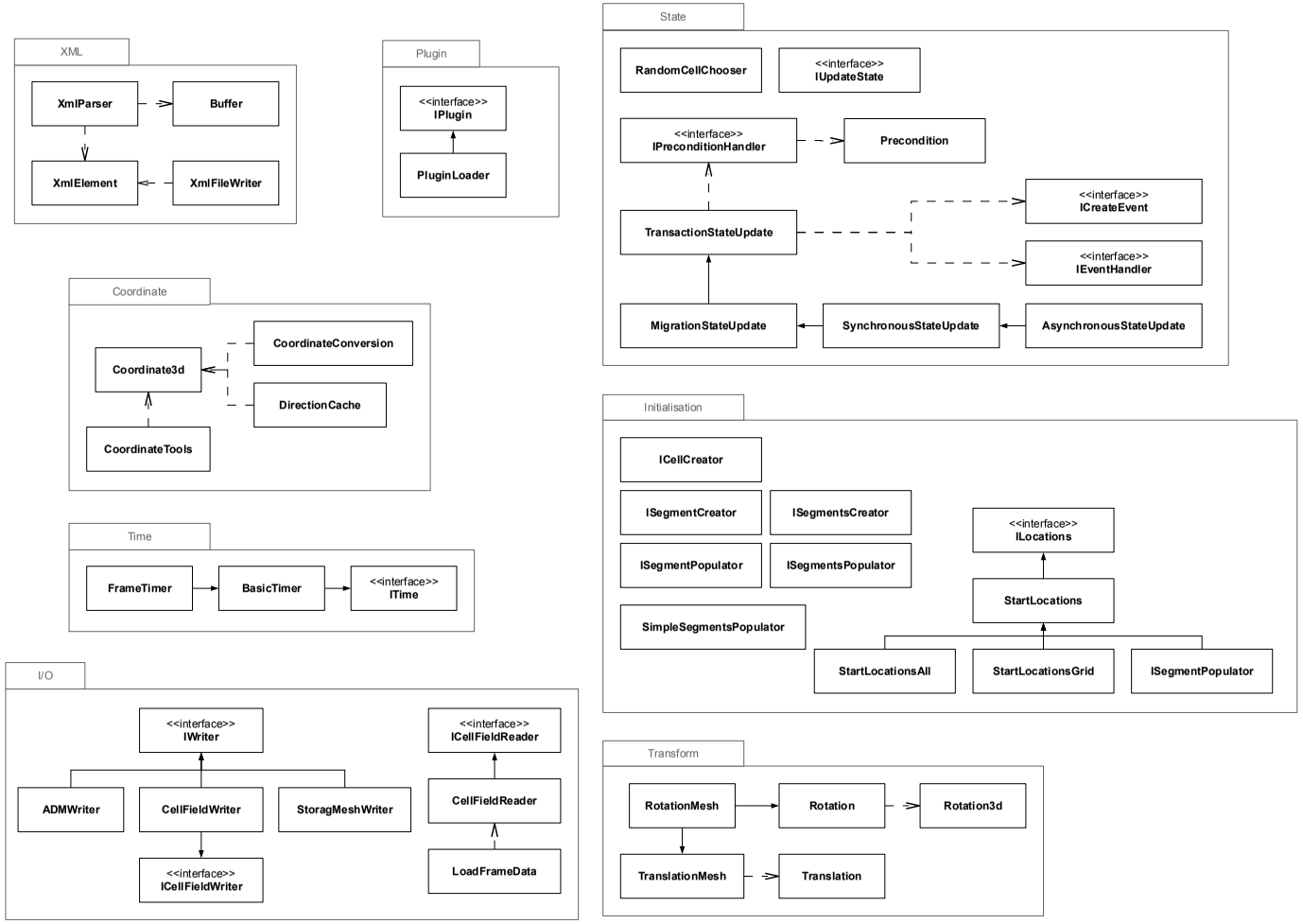


Figure 5.4: Class diagram of the peripheral components of FABCell. The components supplement the core components (see Figure 5.3) and provide additional major functionality such as data input and output and the plug-in framework.

technical description of the implementation. Readers who are interested in the C++ implementation of FABCell can examine the source code which is provided on the CD-ROM accompanying this thesis (see Appendix F).

Agents. Models are composed of a large number of interacting agents, each of which can have an individual set of rules. A FABCell agent is an abstract, independent entity that is capable of performing an arbitrary task. Agents have multiple uses within FABCell and must be further refined into a specific type of entity, such as a *Cell*, to perform a specific role.

Cell. The *Cell* classes provide a way to represent biological cells in the system. Cells are extensions of agents which exist in a real coordinate space within a virtual three-dimensional environment. All cells are derived from *BasicCell*, which provides the minimum functionality in order for a cell to be included in a simulation. Cells are composed of a set of segments each of which is composed of a set of positions (either lattice nodes or real coordinates) in an ordered hierarchy: cell → segment → segment nodes. Operations at the cell level affect all segments and, consequently, all the positions occupied by the cell. Operations at the segment level only affect the physical shape of a particular segment. This flexible design supports different resolutions at which to model a cell. Conventional cellular automata can be simulated by only considering the cell level. More complex cellular agents, where the shape of the cell is important, can be modelled at the segment level. Differentiating a cell into distinct segments is useful for modelling localisation within cells. Each cell maintains its structure internally using a graph (see Figure 5.5). This incurs a small memory penalty, but on a lattice model it provides a very fast mechanism to determine which nodes a cell is occupying and subsequently which cells are its neighbours.

Space. FABCell has two different representations of space: *environment space* and *cell space*. Each simulation is run within a finite simulation volume that maintains a real coordinate system and lattice system describing the space. All cells must be registered within the *environment space*. Cells also exist within their own *cell space* on an infinite

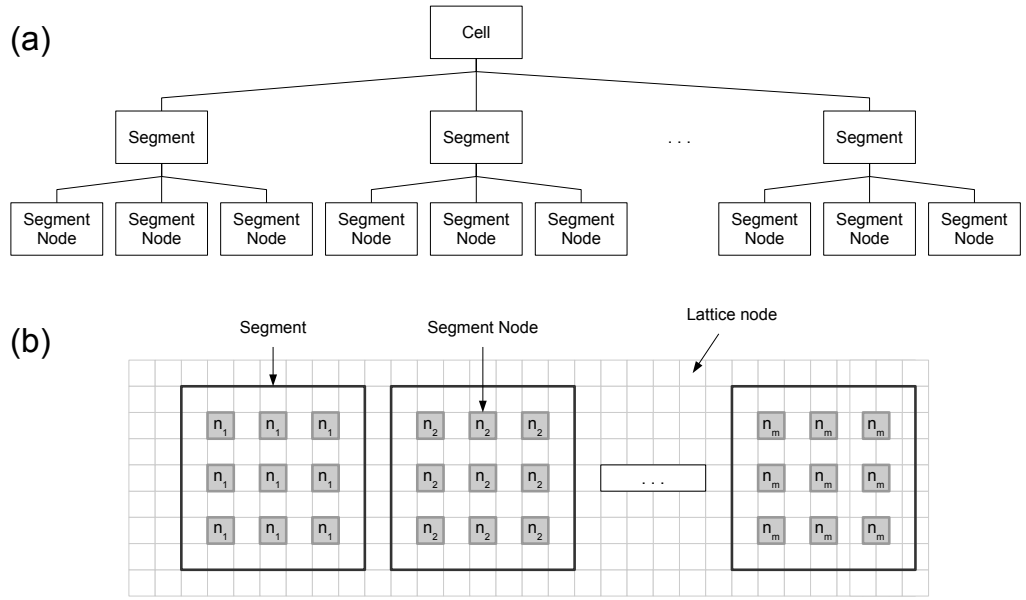


Figure 5.5: Cell representation within FABCell. (a) All cells are composed of a set of segments each of which is composed of a set of nodes defining its topology in the environment. This hierarchy is represented internally using a graph to keep track of the cell. For a cellular automata, cells will have simple trees consisting of one cell, one segment and one node. (b) The topology graph is used to map a cell onto a lattice where each n_x is a node of segment x and a set of segments collectively represent a cell.

plane with each call having a position $p \in \mathbb{R}^3$ and an address $a \in \mathbb{Z}^3$. Positions and addresses can be mapped to the lattice space. Manipulations such as rotations and translations can be performed in the *cell space* before the cell is mapped into the *simulation space*. Having these two spacial systems makes dealing with periodic boundary conditions more straightforward since whilst it might appear a cell splits into two as it moves out of the simulation volume at one edge and moves in at another edge, in the *cell space*, the volume it occupies remain contiguous (see Figure 5.6).

Lattice. Lattice based simulations can be carried out on either a three-dimensional square lattice, three-dimensional hexagonal lattice or three-dimensional triangle lattice. Lattices are used extensively to keep track of objects within a simulation.

Occupancy. The lattice is extended by a set of classes to allow cells to be registered at lattice points. Each node stores a channels and each channel stores b cells. Implementations for single and multiple channels (the

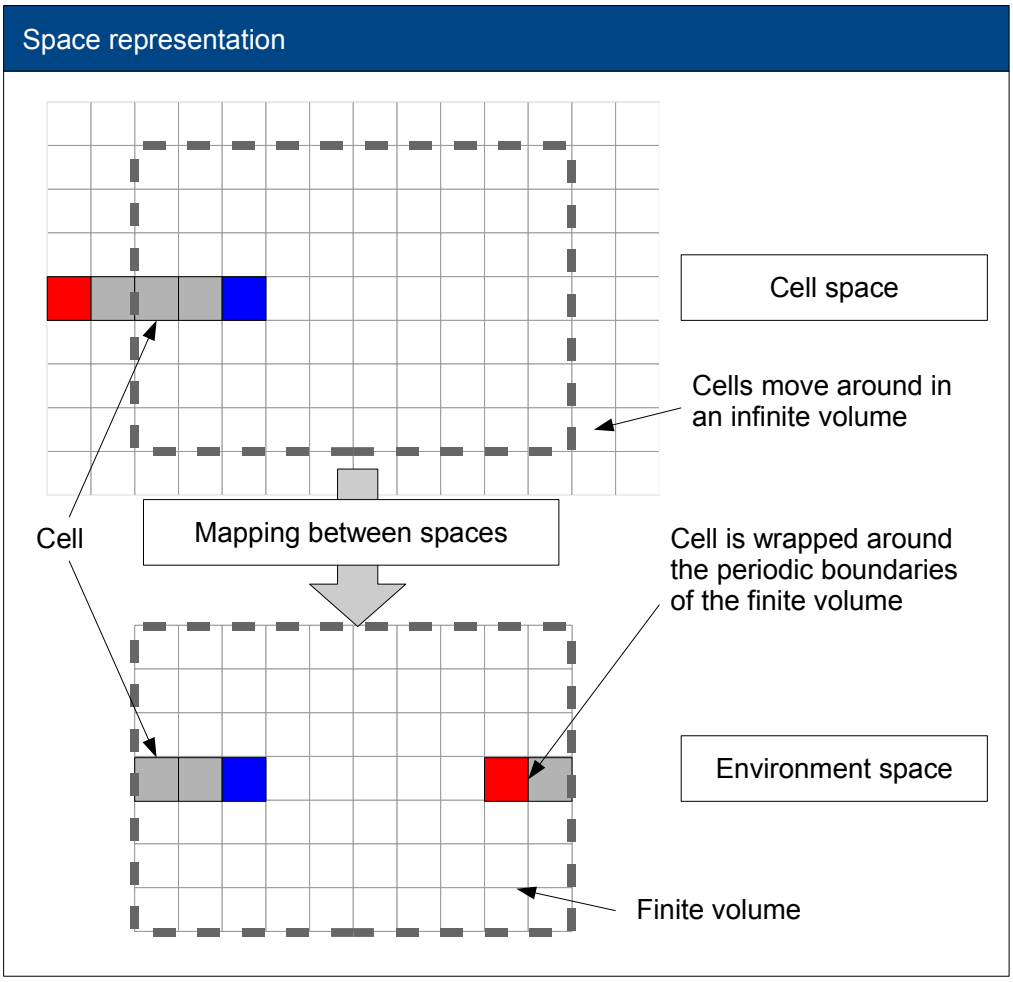


Figure 5.6: Space representation within FABCell. FABCell maintains two space systems: *cell space* and *environment space*. Each cells exist in its own infinite space which can be mapped onto the *environment space*.

SingleChannelOccupants and *MultipleChannelOccupants* classes) and single and multiple occupants (the *SingleOccupants* and *MultipleOccupants* classes) are provided.

Transform. Cells are considered to be physical entities in the environment. Therefore FABCell provides the functionality to rotate, translate and manipulate cells in a three-dimensional space.

5.4.2 The interaction step

Each model is updated by a set of *StateUpdater* objects. *StateUpdater*s update a specific part of the model, for example a cell's position on a lattice. Users can create their own bespoke updaters, but common ones for dealing with cell interactions, cell migration (both synchronous and asynchronous) and the update of the global environment are provided. The two most important StateUpdaters are *MigrationStateUpdate* and *InterationStateUpdate*. During the migration step, cells are moved around on the lattice. This can be done synchronously where all cells are moved based on the current state (i. e. changes are not reflected until the next iteration), or asynchronously where each cell is moved one at a time so that the changes are observable by other cells.

During the interaction step, cells interact with those around them. In a true agent paradigm, agents can only partially observe their environment and are limited to interacting with a local neighbourhood [73]. FABCell allows each cell to check a local neighbourhood of lattice points to determine which cells are in its vicinity. This process is optimised using a caching function [28], which provides a fast method of determining a cell's neighbours. Each cell has a reference to a *RelativeNeighbourhood* object that stores a set of relative coordinates. The relative neighbourhood is overlaid on the lattice at the cell's centre position to get the actual neighbourhood the cell can observe.

Event Model. Each iteration of a model causes a change in state. In a spatial model there are two primary update mechanisms to cause a state change: internal and spatial. Internally a cell updates its state using the agent model discussed earlier. Spatial updates are handled via the event model. A spatial update is prescribed by a sequence of *Event* objects. Events control exactly what happens to a cell, for example a translation

or rotation or reversal. Events are created using *EventCreators* which are passed to the migration state object. During each iteration, the *MigrationStateUpdate* object will run the event creators for each cell and produce a set of *Events* that will update that cell. There is a corresponding set of *EventHandlers* which process events. This transaction system is used to monitor all of the events being submitted and decide if they should be executed.

5.4.3 Precondition handling

Program correctness is an important topic in theoretical computer science for it is a non-trivial problem to determine whether a program will run and terminate correctly. One of the challenges with formal methods is describing complex object structures mathematically.

Initial modelling experiments with FABCell revealed that on numerous occasions it was necessary to check the model state after an iteration to determine that a number of properties of the model were maintained and if not, to rectify this. Essentially, models required a form of precondition checking, but rather than than showing a correctness of the software it is run-time, dynamic check on the models themselves. A simple example would be a model of moving cells where cells are allowed to stack on top of each other. At some point a cell will move out of stack leaving a gap which must be filled. Likewise there will be a point when a cell wants to move into a stack and requires cells to move out of the way. In both cases a form of precondition is being expressed, namely that in order for a cell to move, cells must be moved out of the way first before it can do so. Rather than using an ad-hoc set of routines to check model conditions, FABCell uses the concept of preconditions to allow model properties to be expressed at an object level and checked at run-time whilst a simulation is running.

Dijkstra [29, 34] formally expressed a (weakest) precondition as follows:

$$wp = C \times \mathbf{A} \rightarrow \mathbf{A} \tag{5.1}$$

$$\mathbf{A} = \langle p, q \rangle \tag{5.2}$$

where C is a program or list of statements to be executed, p is a precondition, q is a post condition and $\mathbf{A}$ is an assertion stating that if p holds, q holds.

$$wp(C, q) = \bigcup \{p \in \mathbf{A} : [p]C[q]\} \tag{5.3}$$

The interpretation of the assertion $wp(C, q)$ is that any valid computation C, when started in any initial state $wp(C, q)$, should lead to a finite computation that ends in state q.

Precondition handling within FABCell is part of the main execution loop (see Figure 5.7). The feature is optional and by default is switched off since not all models require it. System events are handled by *event handlers*. Each event handlers can choose to either process an event or raise a precondition event by calling a precondition checking method before proceeding. The method returns a status code indicating what, if any, preconditions should be met for the event to proceed. If a precondition is raised, execution of the simulation is halted and a precondition event is routed to the precondition oracle which attempts to match a *precondition handler* to the precondition. A precondition handler is allowed to create new events and the new events are placed in the execution stack before the current event. Once the new events have executed, the preconditions of the current event are met and execution resumes normally. In order to prevent an infinite cycle of event creation, events created by precondition handlers are granted a higher privilege than standard events and are not checked for precondition themselves. The precondition system is particularly useful in asynchronous models where it can act as a cascade to deal with the repercussions of each cell making state changes in a timely manner.

5.4.4 Neighbourhoods

Neighbourhoods play an important role with FABCell regardless of model type. All simulations whether off-lattice or on lattice use at least one lattice to keep track of objects since it is much faster to lookup items within a lattice than to do burdensome calculations within an off-lattice environment to locate objects. As an example consider

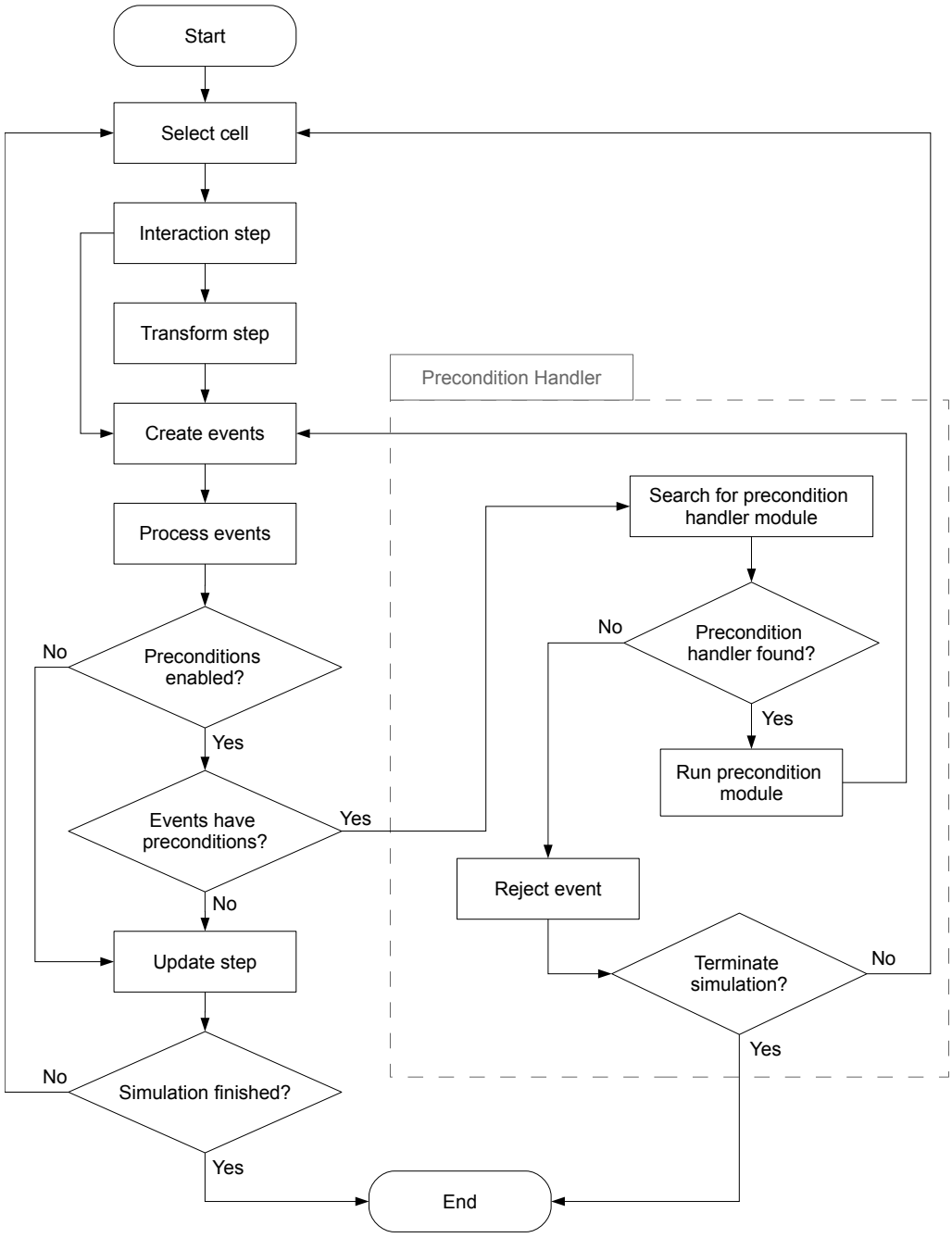


Figure 5.7: The FABCell program loop with precondition handling. Events are queued within the event processor so that they can be validated to ensure the system always remains in a valid state. If an event triggers a violation it is passed to a violation handler which will remedy the problem.

the problem of a cell trying to find its local neighbouring cells. In a lattice model, the local lattice nodes can be explored to find out who the neighbours are, which is computationally relative inexpensive. Without a lattice, $O(n^2)$ comparisons will be required to determine which cells are a particular cell's neighbours. One alternative to using a lattice to speed up comparisons is to use scene graphs which build graphs describing where objects are located relative to each other. This can be faster than a brute force approach; however, there is still a calculation penalty since the graphs must be continually updated as cells move around. A lattice offers fast performance with a small trade-off in accuracy although accuracy can be improved by decreasing the distance between mesh points giving a finer grained mesh.

FABCell supports three basic neighbourhood types (see Figure 5.8): *Moore*, *von Neumann* and *circular*.

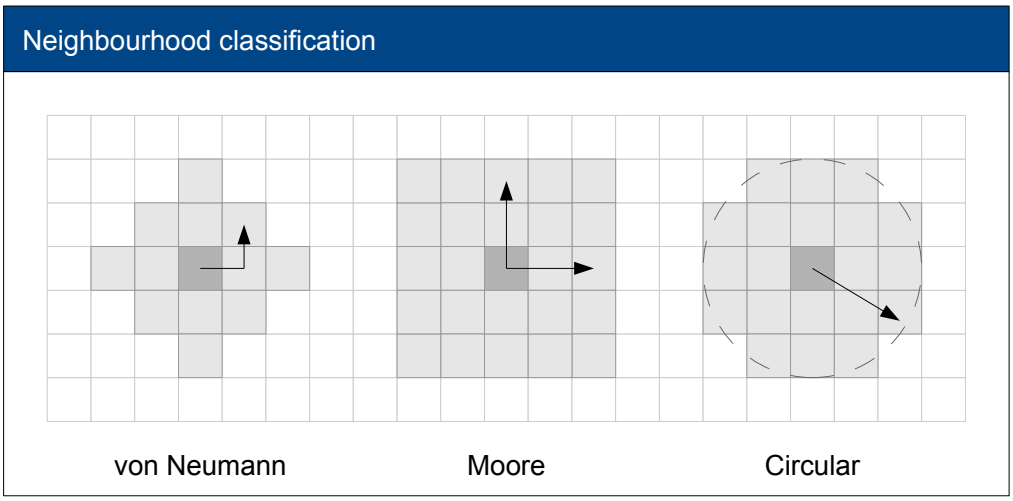


Figure 5.8: Neighbourhood classifications. Dark grey nodes indicate the neighbourhood centre and light grey nodes are the neighbours of the centre node according to a neighbourhood criterion (see main text for inclusion criteria).

Moore neighbourhoods. Comprise all neighbours such that:

$$N_{(x_0,y_0,z_0)} = \{(x,y,z) : |x - x_0| \leq r, |y - y_0| \leq r, |z - z_0| \leq r\} \tag{5.4}$$

von Neumann neighbourhoods. Comprise all neighbours such that:

$$N_{(x_0,y_0,z_0)} = \{(x,y) : |x - x_0| + |y - y_0| + |z - z_0| \leq r\} \tag{5.5}$$

Circular neighbourhoods. Comprise all neighbours such that:

$$N_{(x_0,y_0,z_0)} = \left\{(x,y,z) : \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2} \leq r\right\} \tag{5.6}$$

Creating neighbourhoods has a computation cost; it is very similar to traversing the nodes of a forest. Each time a neighbourhood is required, a search algorithm must explore the local nodes around the given centre node and determine if they meet the neighbourhood criterion. Neighbourhood algorithms often have to be exhaustive and explore a lot of nodes to determine the inclusion set and it is a costly operation to keep traversing the graph of nodes. During a simulation there can be many millions of requests for neighbourhoods which can translates into a significant performance overhead if no form of optimisation is used.

Neighbourhood creation is speeded up by using relative neighbourhoods. Instead of calculating the neighbourhood of a node each time it is requested, the relative offset addresses of the neighbourhood are calculated once at the start of the simulation. The relative offset is row (N_r^r), column (N_c^r) and layer (N_l^r) difference between a neighbourhood node and the node at the centre of the neighbourhood.

$$\begin{aligned} N_r^r &= N_r^n - N_r^c \\ N_c^r &= N_c^n - N_c^c \\ N_l^r &= N_l^n - N_l^c \end{aligned}$$

Neighbourhoods are quickly calculated by adding the offset addresses to a given node address to produce a list of the real neighbourhood nodes. Figure 5.9 describes the relative neighbourhood algorithm used in FABCell.

Relative neighbourhood algorithm

1. Select centre node of lattice.
2. Overlay neighbourhood and determine all nodes within the neighbourhood.
3. Calculate the offset of each node from the centre node.
4. Construct a new neighbourhood where each entry is composed of a node offset.
5. Select node *n* to acquire neighbourhood.
6. For each entry within the offset neighbourhood add it to the node *n* to get a local neighbour node of *n*.

Figure 5.9: Relative neighbourhood algorithm. This algorithm boosts the performance of neighbourhood creation by pre-computing the address offsets of frequently used neighbourhoods. To acquire the actual local neighbourhood of a node, the offsets are added to the node address to give the real addresses of the neighbourhood nodes.

5.4.5 Creating a model

All FABCell models adhere to a specification which dictates the core objects that must be present for the model to run. The libraries provide a catalogue of objects which can be plugged together to form a model environment, or else they can be extended to provide bespoke functionality. An outline of the steps required to create a working model is given in Figure 5.10. Models can be instantiated either directly using C++ or an XML description file and at all stages user defined classes can be used in place of those provided in the FABCell libraries.

5.5 Design patterns

A common approach to software design (and problem solving in general) is to employ a *divide and conquer* strategy. A problem is conceptualised as a set of smaller sub problems which are more tractable so that they can be solved and the results combined into the solution to the actual problem. This approach works well for specific problems but for application development a more structured and ordered approach is

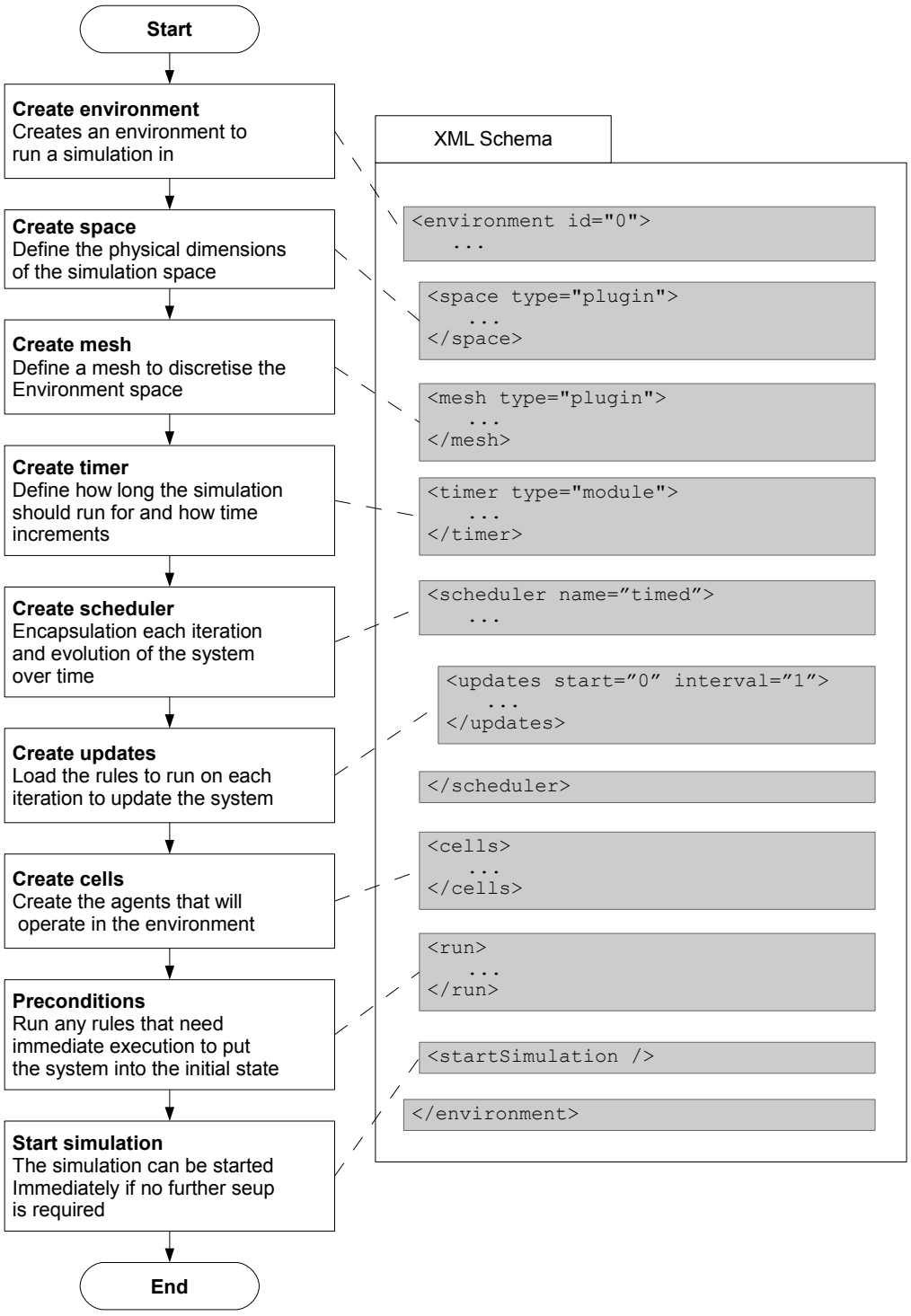


Figure 5.10: Flowchart of the steps required to create a model. Models can be instantiated either directly using C++ or an XML description file.

often required. Divide and conquer works well for specific well defined problems, but in application development the solution is usually more generalised and complex with many (often conflicting) goals and users. An alternative approach is the use of *design patterns* [13].

Design Patterns

Design patterns encapsulate common ways of solving problems using language features together.

Design patterns enforce good coding practice and ensure that code is written for maximum flexibility. By following design patterns a form of divide and conquer is effectively implemented since each project is engineering in the same way so that is conforms to a set of patterns. This section is not an exhaustive review of design patterns; for brevity only those patterns which are pertinent to FABCell are discussed.

Decorator. A decorator provides a way of attaching new state and behaviour to an object dynamically [14]. In C++ this is achieved through the use of `virtual` methods which allow sub classes to override them and provide a reimplementation. Decorators are used extensively in FABCell to provide new functionality. The plug-in system for example is heavily dependent on decorators to add new functionality.

Proxy. A proxy controls the creation of and access to other objects. In FABCell this is enforced with the use of access modifiers on all class members to ensure encapsulation. Where possible every member is `private` and is only exposed to other classes if absolutely necessary. In this way objects and users are prevented from accessing internal data that they do not require access to. This is to ensure system integrity; only the correct modules change system state ensuring consistency.

Factory Method. A factory method creates appropriate objects based on information supplied by the client [17]. Objects have a common interface and are specialisations of

a more generic object. The factory decides what the most appropriate object to return is. In FABCell the viewer uses factory methods to create various viewer modules based on an XML schema specifying how the output should be rendered. Factory methods encapsulate the creation and initialisation of objects (which can be complex) so that the client does not have to know the details and simply specifies the object type.

Singleton. A singleton ensures that there is only ever one instance of a class with a single global access point to that object [17]. In FABCell certain objects such as the random number generator are provided as singletons to prevent multiple instances. Many FABCell objects such as cells and cell segments are given unique identifiers so they can be tracked during a simulation. As an object is created it requests a new identifier from a singleton that controls a global pool of identifiers to ensure no two objects ever get the same identifier.

Iterator. An iterator provides a way of accessing elements of a collection in a sequential fashion without the accessing object/user needing to know how the collection is structured [15]. FABCell makes use of iterators to allow access to collections of objects such as cells. For example, the advantage of iterators is that the underlying collection of cells can be reordered and optimised for specific purposes. In some instances cells are returned in a random order, in others sequentially depending on the model being run, users are presented with a uniform interface to get access to the cells without having to be concerned about how they were ordered.

Observer. An observer enables objects to communicate without knowing the identies of the objects they are communicating with. Objects broadcast a public message which other objects can respond to. In FABCell this is encapsulated in an `event` model. Certain objects are allowed to act as `observers` which wait for other objects to communicate with them. Another set of objects act as `broadcasters`. These objects allow `observers` to register with them and can then subsequently notify them of system changes. One such use of this is notifying cell iterator caches that the number of cells has changed and they need to reorder their internal lists.

Interpreter. An interpreter interprets instructions written in a language defined for a specific purpose [16]. In FABCell this is via the extensive use of XML. All plug-ins must support an XML interface that allows them to be created from an XML declarative that specifies all initial parameters of the object. A FABCell model can be designed and initiated almost entirely through an a single XML file making it easier for non-programmers to use the system.

5.6 Parallelism

At the forefront of modern high performance computing is the use of parallel processing to execute multiple tasks at once. A conventional processor will process instructions sequentially. By parallelism we typically mean the division of a computation into multiple parts which can then be run independently and concurrently. The topic has been actively researched for some thirty years and is a feature of almost all supercomputers and high performance workstations. Modern CPUs for desktop computers made by both Intel® and Advanced Micro Devices™ (two of the world's largest microprocessor manufacturers who together supply over 90 % of the world's microprocessor) contain multiple cores[5] effectively allowing everyone access to relatively low cost, commodity hardware that can run multiple processes at once.

Applications such as simulation software are often written to exploit the inherent parallelism in the hardware to boost performance. Simulation tools that take advantage of parallel processing allows more complex and sophisticated models to be created and can greatly reduce the amount of time required to run large simulations. FABCell is a high performance modelling environment, which could potentially simulate many hundreds or thousands of cells, so this section looks at how FABCell has been designed to exploit parallelism.

5.6.1 Implementing concurrency in C++

Parallel architectures are often described in terms of Flynn's Taxonomy [117] which classifies architectures into four groups:

[5] A multi-core processor combines two or more independent CPUs into a single die (integrated circuit).

SISD Single instruction stream, single data stream.

SIMD Single instruction stream, multiple data streams.

MISD Multiple instruction streams, single data stream.

MIMD Multiple instruction streams, multiple data streams.

MIMD is the most relevant to parallel architectures since it describes systems which can execute multiple different instructions (possibly on different processors) and read and write to multiple data sources. MIMD can be further refined into two subcategories:

SMP Symmetric multiprocessors consisting of a few processors with shared memory, which communicate via memory.

MPP Massively parallel processors consisting of many processors with distributed memory, which communicate via a network.

From an implementation perspective, writing C++ software for these types of architecture is generally achieved through the use of specialised APIs. Software must be written correctly to exploit concurrency which is more difficult than procedural programming; greater effort must be taken over the granularity of the design so that components can be safely run in parallel.

MPP architectures. The Message Passing Interface (MPI) [131] is commonly used on MPP architectures. A program is compiled once and then distributed to a number of node processors on a network which each execute the program. The programmer must specify what each processor executes (typically using a case statement linked to the node identifier so that the program branches and runs differently on each node) and also what is communicated between nodes. One node is typically to be the master node which collates the results of the other nodes.

SMP architectures. The first approach is to use the native threading mechanisms of the development language. In C++ this involves the use of the pthreads [9], which provides a way to manually create and control threads. This method has the most potential for error since it relies on the designer ensuring everything is implemented correctly. OpenMP [19] is commonly used on SMP architectures. A program is compiled once and then distributed to a number of node processors on a network which each execute the program. The programmer must specify what each processor executes (typically using a case statement linked to the node identifier so that the program branches and runs differently on each node) and also what is communicated between nodes. One node is typically to be the master node which collates the results of the other nodes.

FABCell was designed with an SMP in mind. Models typically consist of a large number of interacting entities within a virtual environment. It requires a lot of memory to maintain the system state so it would be undesirable to replicate each model across every node in a MPP system. Synchronisation also becomes an issue since each copy of the model must be notified of the changes made on every other copy and ensuring that the states remain consistent can be difficult. The network in an MPP is usually the slowest component of the system and a bottleneck. Node synchronisation requires a lot of network traffic between nodes so any performance gain achieved through parallelism may be lost through latency in communication.

FABCell is designed to work on conventional workstations, so consideration had to be given to allowing it to run on a single processor. MPI and pthreads both require the software to be written explicitly to exploit parallel processing rendering it unsuitable for running on a single processor or systems that do not support the APIs. OpenMP allows code to be written that can be much more easily tailored to a particular architecture and for this reason and the limitations of MPI and pthreads outlined it was designed to adapt OpenMP in FABCell.

OpenMP is based upon the existence of multiple threads in the shared memory programming paradigm. A shared memory process consists of multiple threads. OpenMP uses a *Fork - Join* model (see Figure 5.11) to accomplish parallelism:

- A program begin as a single process called the *master thread*. The master thread executes sequentially until the a parallel region construct is encountered.
- At a parallel region, a team of threads is created and each thread executes the statements in the parallel region in parallel with the other threads in the team.
- At the end of the parallel region is a join region to synchronise the threads. Each thread waits at the join region until all of the other threads in the team have finished. All of the threads then terminate together referring control back to the master thread, which resumes sequential execution.

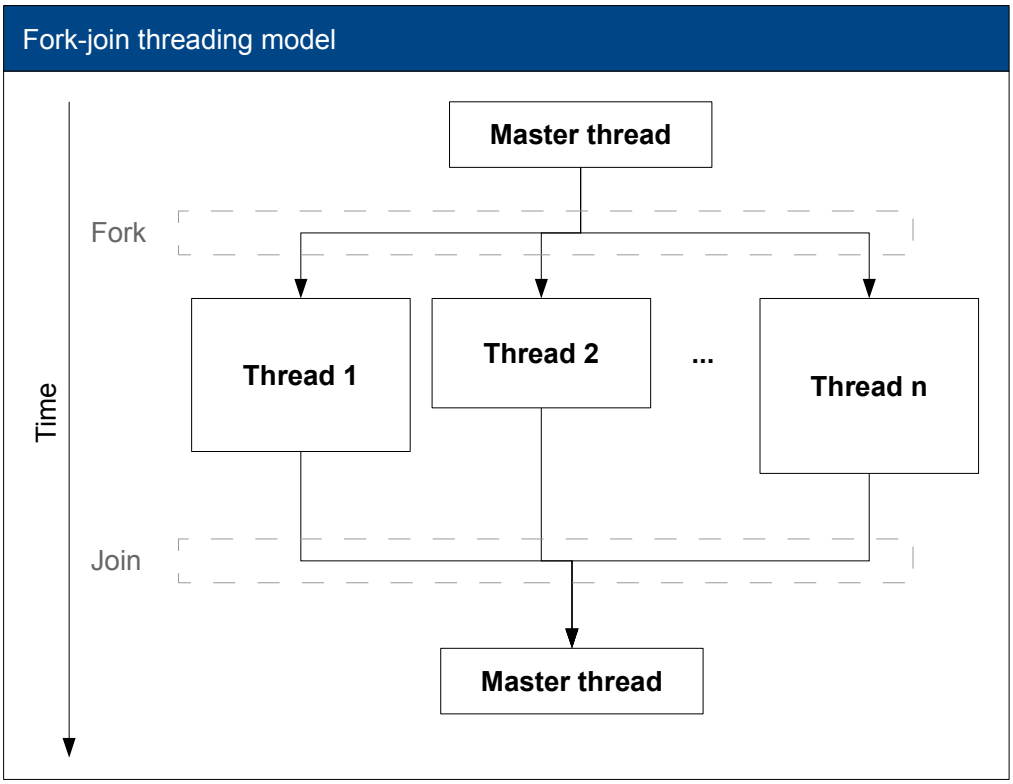


Figure 5.11: Fork model of parallelism used by FABCell.

OpenMP makes use of the C/C++ pragma[6] directive to indicate where to parallelise code. It is not possible to discuss OpenMP in its entirety; however, an overview of

[6] A pragma directive provides additional information to the compiler, beyond what is conveyed in the language itself. It offers a way for each compiler to offer machine specific features while retaining overall compatibility with C/C++. If the compiler does not recognise the pragma, it is ignored.

how it is used will be provided. Two of the most important features of the specification are the ability to create threads and the ability to create *critical regions*[7] which are often used to protect shared data from race condition such as two threads attempting to write data to the same location at the same time.

`parallel` **pragma.** Indicates that a block of code should be duplicated and run as a specified number of independent threads.

```
#pragma omp parallel
```

where `omp` indicates it is an OpenMP pragma, `parallel` indicates proceeding code should be threaded.

`critical` **pragma.** Indicates that a block of code should be duplicated and run as a specified number of independent threads.

```
#pragma omp critical (name)
```

where `omp` indicates it is an OpenMP pragma, `critical` indicates proceeding code has a lock called `name` assigned to it allowing only one thread at a time to execute it.

Figure 5.12 is a fragment from the FABCell code base showing a real world example of how OpenMP can be used. A Hamiltonian function calculates the sum of a number of independent energy terms. This is parallelised by using the `parallel` pragma to assign each energy term to its own thread so the time taken to calculate the total energy is the time it takes for the slowest energy term to compute rather than the time taken for all the energy terms to compute. the `critical` pragma is used to ensure the summation of `energy` remains consistent. Note the use of the C/C++ pre-processor directive

```
#ifdef DEF_FABCELL_PARALLEL_OPENMP
```

which allows the code to be written to execute procedurally or in parallel depending on requirements. In this way FABCell can be compiled to run on a variety of architectures.

[7] Specifies that although a code block may be shared by all threads, it will only executed by one thread at a time. If a thread is currently executing inside a critical region and another thread reaches that critical region and attempts to execute it, it will block until the first thread exits that critical region.

FABCell OpenMP parallel enabled code

```
while (iter != energyFunctions.end()) {

#ifdef DEF_FABCELL_PARALLEL_OPENMP
#pragma omp parallel private (execute, function, change)
  {
#pragma omp critical (hamiltonian_summation_lock_1)
    {
      if (iter != energyFunctions.end()) {
#endif

        function = *iter;
        ++iter;

#ifdef DEF_FABCELL_PARALLEL_OPENMP
        execute = true;
      }
      else {
        execute = false;
      }
    }
#endif

#ifdef DEF_FABCELL_PARALLEL_OPENMP
    if (execute) {
#endif
      change = function->Energy(expandEvent);

#ifdef DEF_FABCELL_PARALLEL_OPENMP
#pragma omp critical (hamiltonian_summation_lock_2)
      {
#endif

        energy += change;

#ifdef DEF_FABCELL_PARALLEL_OPENMP
      }

    }
#endif
}
```

1. Fork/create threads
2. Enter critical region
3. Leave critical region
4. Enter critical region
5. Leave critical region
6. Join threads

Figure 5.12: FABCell OpenMP parallel enabled code. The example shows the threaded computation of the sum of a number of energy terms where each term is computed in its own thread. The use of the `#ifdef DEF_FABCELL_PARALLEL_OPENMP` pre-processor directive allows both parallel and procedural code to be written together so it can be executed using either method depending on whether the architecture supports parallel OpenMP.

5.6.2 A formal expression of concurrency

The `#pragma omp parallel` directive is a concrete implementation of Dijkstra's `parbegin; ...; parend` construct for parallel execution [33]:

`begin` s_1`; parbegin` s_2`;` $s_3 \cdots s_{n-1}$`; parend;` s_n`; end`

`parbegin` s_2`;` $s_3 \cdots s_{n-1}$`; parend` defines a parallel compound in which all of the constituent statements s_2 to s_{n-1} are executed in parallel. The execution of the compound is completed after all of the constituent statements have executed and only then will the execution of s_n be initiated. In the above statement, s_1 represents the master thread which forks at `parbegin` and runs s_2 to s_{n-1} concurrently as threads. s_2 to s_{n-1} join at `parend` and then the master thread (s_n) continues execution.

The `#pragma omp critical` directive is an implementation of a critical region. This is a section of code that guarantees at most one thread will have access to the resource at any one time. Essentially it behaves like a binary semaphore granting an access token to the competing threads. Whichever thread gets the token is allowed to enter the section and access the resource. Once the thread has finished the critical region it relinquishes the token so that another thread may enter. A semaphore can be described formally using the Calculus of Communicating Systems (CCS) [113], an abstract mathematical theory of concurrency.

A critical region is equivalent to a one token semaphore which can be defined:

$$\mathtt{Sem} \stackrel{\text{def}}{=} \mathtt{get.put.Sem} \tag{5.7}$$

$$\mathtt{Critical} \stackrel{\text{def}}{=} \mathtt{Sem}[\mathtt{getc/get}, \mathtt{putc/put}] \tag{5.8}$$

$$\mathtt{Thread} \stackrel{\text{def}}{=} \overline{\mathtt{getc}}.(\mathtt{a}_1.\mathtt{a}_2 \cdots \mathtt{a}_n).\overline{\mathtt{putc}}.\mathtt{Thread} \tag{5.9}$$

A thread must have access to $\overline{\mathtt{getc}}$ in order to enter the critical region and performs actions $\mathtt{a}_1$ to $\mathtt{a}_n$.

The system is formed from the composition of multiple threads and a critical region agent.

$$(\texttt{Thread}_1|\texttt{Thread}_2|\cdots|\texttt{Thread}_n|\texttt{Critical})\backslash\{\texttt{getc},\texttt{putc}\} \tag{5.10}$$

5.7 Creating a user interface

Up to this point, the design has focussed almost exclusively on the application from a technical and development rationale; this is not the concern of the typical user who will simply want to use the tool and meaningful data from it. Indeed an explicit requirement from Section 5.2 is for users to be abstracted from the internal workings of FABCell as much as possible.

A well designed user interface improves the quality of software [180] by providing some of the following benefits:

Efficiency A consistent interface allows users to quickly learn and remember how to use an application to perform tasks efficiently.

Reduced errors Users are insulated from the complex internal workings so are less likely to make errors from misuse of the software.

Improved acceptance Users tend to prefer systems which are well designed so they can quickly find out how to perform a task.

Reduced training Users can create a proper learning model of how to use the software and a consistent design promotes reinforced learning; concepts from one part of the system apply in other areas as well. By allowing different classes of problems to be investigated within the same framework, FABCell reinforces its design paradigm. There are a finite number of extremely similar work flows required to get any model working.

FABCell was designed as a tool for research with an emphasis on users in biological research fields. The user base is an important consideration since it dictates how user friendly software should be. Ideally all programs should be designed to be easy to use and intuitive but in reality this is often not the case, particularly as software becomes more technical and specialised, there is usually an increased assumption about the

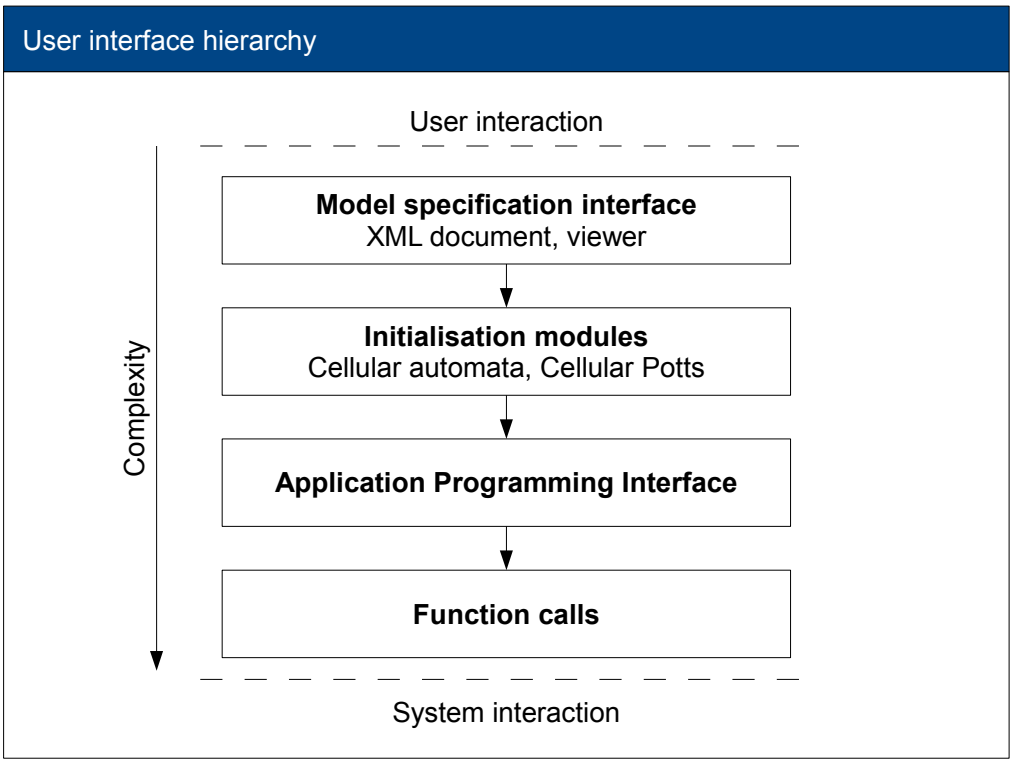


Figure 5.13: The FABCell user interface hierarchy. Each level provides an abstraction of the levels below it with an increasing ease of use but decreasing flexibility.

technical proficiency of the user. FABCell was written completely in C++ necessitating that its users understand how to write code to make use of it. This is an immediate barrier for users particularly those who have neither the time nor inclination to learn how to code. Indeed as a research tool, FABCell should largely conceal its inner workings from users and place minimal requirements on them to understand low level design features. To accomplish this, the software is designed in a hierarchical top-down fashion (see Figure 5.13). In abstract terms, FABCell is a set of functions (albeit very large) which perform specific tasks. A subset of the functions are designed for users so that they can interact with the system. These public functions are referred to as the *Application Programming Interface* (API). A simulation is a list of API functions called procedurally, which together create a model. FABCell has hundreds of API functions available making it potentially complicated for users to understand how to use them properly. To reduce the complexity, at each level in the hierarchy, functions from lower levels are aggregated and encapsulated into new functions which are designed to

make them easier to user. The module level provides functions to perform common tasks such as creating and initialising cells. There is a trade-off in flexibility; higher level API calls restrict what users can do, but they make tasks easier by shielding users from implementation details and also ensure the API is used correctly. Since all levels of the hierarchy are available to users, they can make a choice of what level to interface with FABCell at depending on whether an existing module provides the functionality or a bespoke solution is required.

5.7.1 Model specification interface

The highest level of the interface hierarchy is the model specification level. This is highly abstracted away from C++ code and allows a simulation to be described in terms of a model specification file. A model specification file is an extensible markup language (XML) document[8] containing directives describing a model (for an example document, see Figure 5.14). It is the most user friendly way of implementing a model, although it sacrifices some of the flexibility that is inherent when developing in C++. A model can be described entirely using a model specification file depending on its complexity. Appendix E lists a full implementation of a model of Wolfram's rule 110 cellular automaton [194], which can be started and run without users having to do any programming.

Models that can be described purely using the XML model specification document can be run using the small reusable C++ program listed in Figure 5.15. It is essentially a skeleton program that makes successive calls to the XML API to create all of the necessary simulation objects.

The FABCell XML schema is designed specifically for creating fabcell models. Recently there was been a concerted effort by projects such the Systems Biology Markup Language (SBML) [143] and CellML [130] to create XML schemas for describing biological events. These schemas concentrate on describing networks and reaction kinetics within a cell so they are not suitable for describing FABCell models. However, FABCell

[8] XML is a general-purpose specification for creating custom markup languages to create and share structured data. Data is surrounded by markup elements of the form `<element>...</element>` which describe the data [55].

XML model specification excerpt

```
<environment id="0">
  <space type="plugin">
    <plugin name="space" lib="libfabcell_engine_space.so"
call="SpacePluginCreate">
      <boundary>
        <x>none</x>
        <y>none</y>
        <z>none</z>
      </boundary>
      <dimensions>
        <x>250</x>
        <y>250</y>
        <z>40</z>
      </dimensions>
    </plugin>
  </space>

  <mesh type="plugin">
    <plugin name="square_mesh"
lib="libfabcell_plugins_square_mesh.so"
call="SquareMeshPluginCreate">
...
```

Figure 5.14: Excerpt of XML model specification.

is designed with extensibility in mind and these methodologies can be incorporated into FABCell components and can be used to describe reactions inside agents/cells. Figure 5.16 in Section 5.8 illustrates how CellML code can be passed into a plug-in for further processing.

5.8 Plug-in Framework

FABCell is designed to be extensible using a plug-in framework. It was decided from the outset that FABCell should be extensible because it was unlikely that every possible feature could be conceived during the initial development and a way of adding new features without requiring drastic changes to the existing software was sought. Extensibility in software, certainly in an OO paradigm, is often achieved through interfaces, declarations of virtual functions prescribing how functions and objects should operate. This provides an efficient mechanism for creating objects and ensuring they obey a consistent specification; however, it is a methodology best suited to design time engineering, when code is initially written. Once the software has been created, although

Code to instantiate a cellular automata simulation

```
int main()
{
  fabcell_modules_io_output_binary::ModelData modelData;

  fabcell_engine_cell::LatticeCells *cells;

  boost::shared_ptr<fabcell_xml::XmlElement> settings =
    fabcell_modules_io_input_settings_xml::LoadSettings::Load();

  boost::shared_ptr<fabcell_engine_lattice_shared::Environment>
   Environment = CreateEnvironment::Create(settings, false);

   // create the cells
  CreateCells::Create(settings,
                      environment,
                      environment->GetTimer());

  // create the rules to update the system
  AddUpdates(settings, environment);

  // run some essential plugins
  LoadRunnablePlugins::Run(settings, Environment);

  // write some binary data about the state of the system
  modelData.Write(environment);

  // run the simulation
  fabcell_engine_simulation::Simulation simulation(environment);

  // run the simulation
  simulation.StartSim();

  return 0;
}
```

Figure 5.15: C++ code snippet to create a cellular automaton. A large proportional of FABCell functionality is encapsulated in wrapper functions which reduce the amount of effort required by a user to start a simulation. A cellular automaton can be started with a small, reusable ten line program and an XML model specification file.

it is straight forward to add new classes, in the case of C++ it requires the designer to integrate the code into the current solution and recompile the new code into the code base. A large project may require significant time and effort to be recompiled, especially if there are multiple additions each requiring their own recompilation.

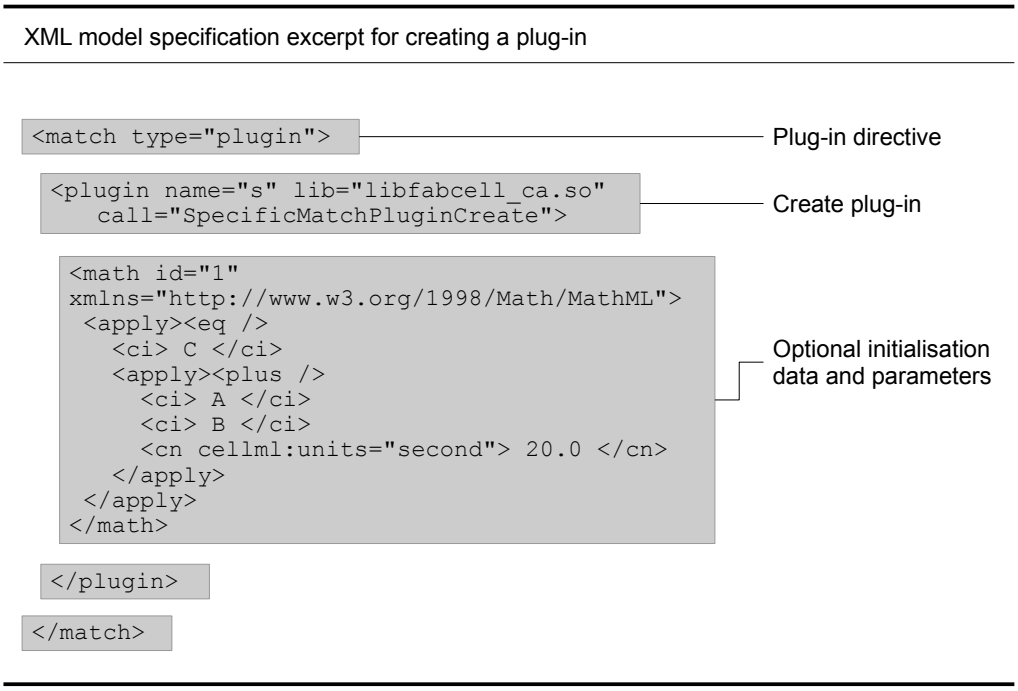


Figure 5.16: Loading a plug-in using XML. Plug-ins can be loaded dynamically with an XML model specification document using the plug-in directive. Plug-ins can be passed arbitrary XML specifying parameters and initial values if necessary, in this case the plug-in is receiving some CellML code for processing.

Plug-ins can be loaded using a simple XML based plug-in declarative (see Section 5.7) to load models and data into a FABCell simulation. Certain simulations can be instantiated solely through the use of linking plug-ins together. Figure 5.16 illustrates the mechanism for using a plug-in with the XML interface. It begins with the plug-in directive `type="plugin"` to tell FABCell that the required system object will be loaded from a plug-in rather than an existing FABCell library. The second direction, `<plugin ...>` has three attributes: `name`, `lib` and `call`. `name` specifies the name of the plug-in for reference, `lib` is a file name specifying where the library containing the plug-in is located and `call` specifies the bootstrap function that should be called within the library to create the plug-in object. Any XML between the `<plugin ...>...</plugin>`

tags is passed to the bootstrap function to allow parameters and extra data to be passed to the object creator. Each of the primary simulation directives in Figure 5.10 allows the `plugin` direction to be used in place of the built in system calls and is the method used to allow users to replace all the major system components as necessary.

5.8.1 Object caches

Complementary to the plug-in framework is the object cache. This allows objects to be loaded into a repository for reuse. It was designed to make it easier to mix C++ code and XML schemas together. Objects can be created within a code block

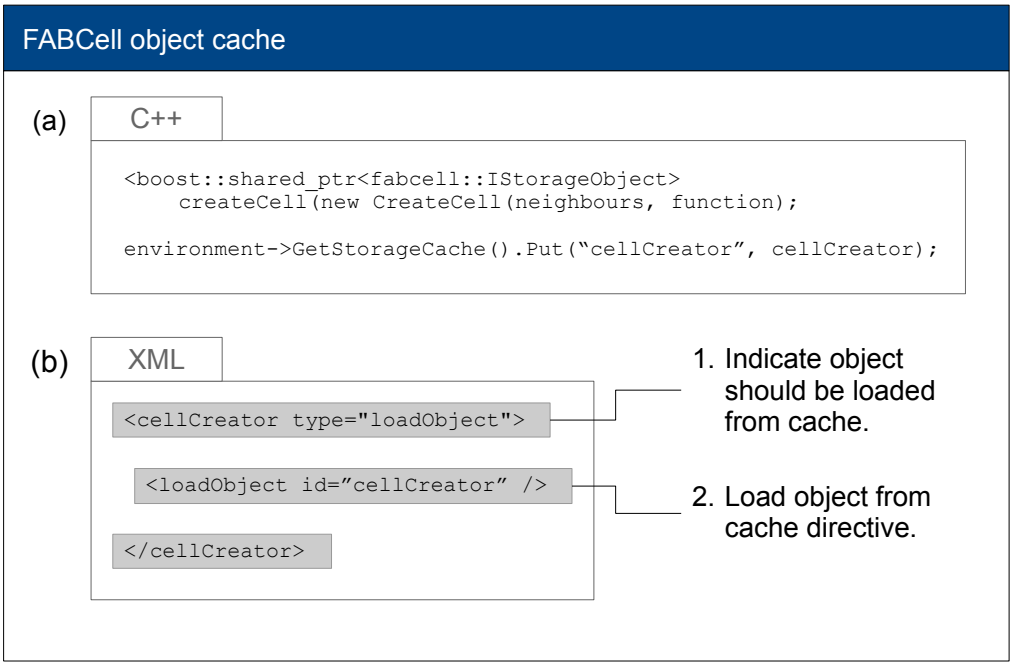


Figure 5.17: FABCell supports object caching for later reuse. This feature allows custom objects to be created and then loaded via an XML schema. (a) Example C++ code to create a custom *CellCreator* object and then store in the object cache as "cellcreator". (b) The "cellcreator" object can be defined for use in the XML schema using the *loadObject* directive and specifying the identifier of the object to use, in this case "cellcreator".

A common design pattern of C++ programs is the *declaration and definition* model. Variables and objects are first *declared* so that the compiler associates them with a certain type, without allocating storage for them and then at a later stage they can be *defined* where storage space is allocated, the object is created and put into use.

The object cache was designed to extend this paradigm with the declaration role

being fulfilled by statements within a C++ code block and the definition role being handled by XML which directs FABCell to allocate an object for use. Custom objects and variables can still be declared within a simulation but can be controlled by the XML schema. Users are given a choice of whether to start a simulation purely through code invocation, pure XML or a mixture of the two. Using pure XML abstracts users from the implementation making it a more suitable choice for non-programmers.

5.9 Visualisation

The core FABCell simulation engine generates textual and binary data about the state of the system. To start to perform analysis on the data, a complementary viewing application was developed to allow users to visualise their models. The FABCell viewer inherits functionality from the main FABCell code base and uses the same extensible plug-in model to allow it to be customised.

Table 5.2: FABCell viewer object types.

Object type	Description
Data processing	Objects in charge of reading in data and converting into useful data structures.
Display modules	Objects which use *Data processors* to create data structures representing every cell within an simulation iteration. Certain display modules can directly render data as well using the *Shapes* objects.
Shapes	Primitive objects which render shapes using OpenGL, for example the FABCell library contains shapes for rendering cubes, hexagons and rectangles.
View modules	Specialised renders with a common interface. In cases where *Display modules* support multiple views, rendering can be deferred to *view modules* to promote object reuse.

Table 5.2 lists the four main object types the viewer uses to render FABCell data visually. Model data is loaded into an object representation of the data. The viewer typically reads from binary files although custom data readers can be added if necessary. *View* objects visualise data in a particular way. A generic graphics pipeline performs a series of steps to translate raw data from a simulation into an on screen object. Fi-

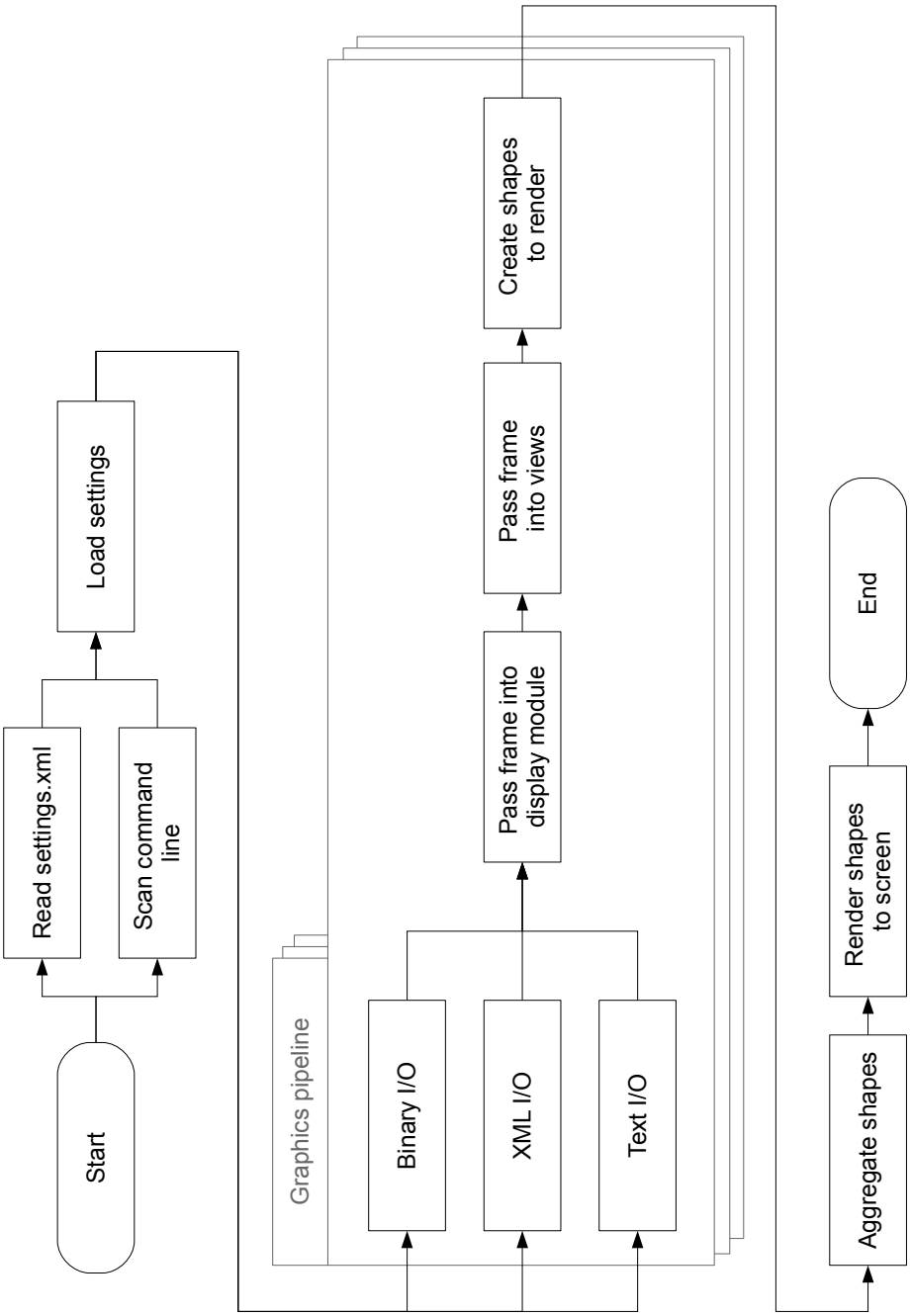


Figure 5.18: FABCell Viewer graphics pipeline. The simulation output is a composite of many different rendering elements

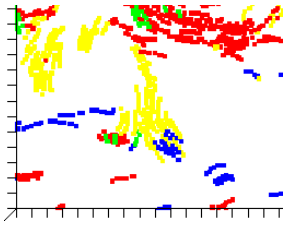
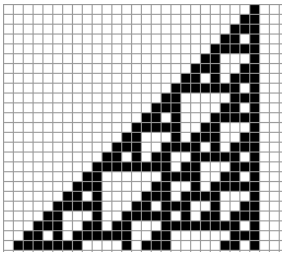
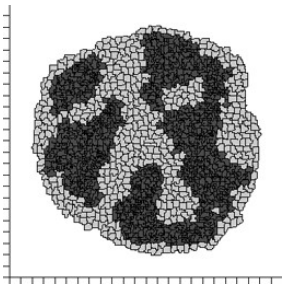
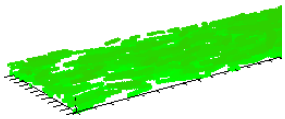
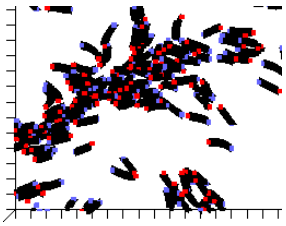
Name	View	Description
Angular Domain Mapping		Cells are coloured by their orientation.
Cellular Automaton		Renders the output of conventional cellular automata by colouring cells by their current state.
Cellular Potts Model		For the output of small cell Cellular Potts Model simulations. Colours each cell by its type.
Density		Plots agent locations as a density map.
Layers		Colours cells by their height above the ground in a 3D environment.
Segment		Individual cell segments are coloured by type so the head and tail are discernible.

Table 5.3: A list of the FABCell viewer display modules and how they render simulation output. All modules offer customisation of their output and new modules can be added if the library functionality does meet the needs of the user. The Angular Domain Mapping (ADM) module is based on ADM as described by Pelling et al. (126)

gure 5.18 shows a diagram of the steps (or pipeline) required to go from raw simulation data to a visual representation on screen. When the viewer is run it consults a settings file called `viewer.xml` which dictates all the modules it should load. Once settings have been read, the viewer runs a number of *display* modules which are in charge of rendering output. Each *display* module loads simulation data either from file or a system cache into an object representation of the data. Once the simulation data has been turned into an abstract representation of the iteration, it is passed as a parameter to a number of *view* modules. A *view* module generates a series of primitive *shape* objects based on the simulation data. These are aggregated together and finally sent to the renderer to be turned into OpenGL commands for displaying on screen. Table 5.3 lists the primary ways the viewer can render cell data.

5.10 Testing

An important part of the software development process is testing. A systematic way of testing software is required in order to verify the software works to specification [181]. Testing has two main purposes [159]: to demonstrate to the developer and customer that the software meets its requirements and to discover faults or defects where the software does not conform to its specification.

FABCell was tested using the process outlined in Figure 5.19. This is an iterative method that repeated compares the output of the system with known test cases to check for errors. The *oracle* (either a user or automated program) matches test data to output data. FABCell was tested as a whole by implementing published models and running them to verify that the output matched published data.

More specific testing was carried out using unit testing. This verifies individual components by looking for faults in their output [159]. Test cases were designed for each component to test how they responded with valid and invalid data.

The majority of testing was carried out using both modified classes to verify input and output using the GNU Project Debugger (gdb) for implementation specific errors, such as invalid pointer addressing. The separation of the system into multiple libraries also facilitated testing since a particular library could be tested individually and it was

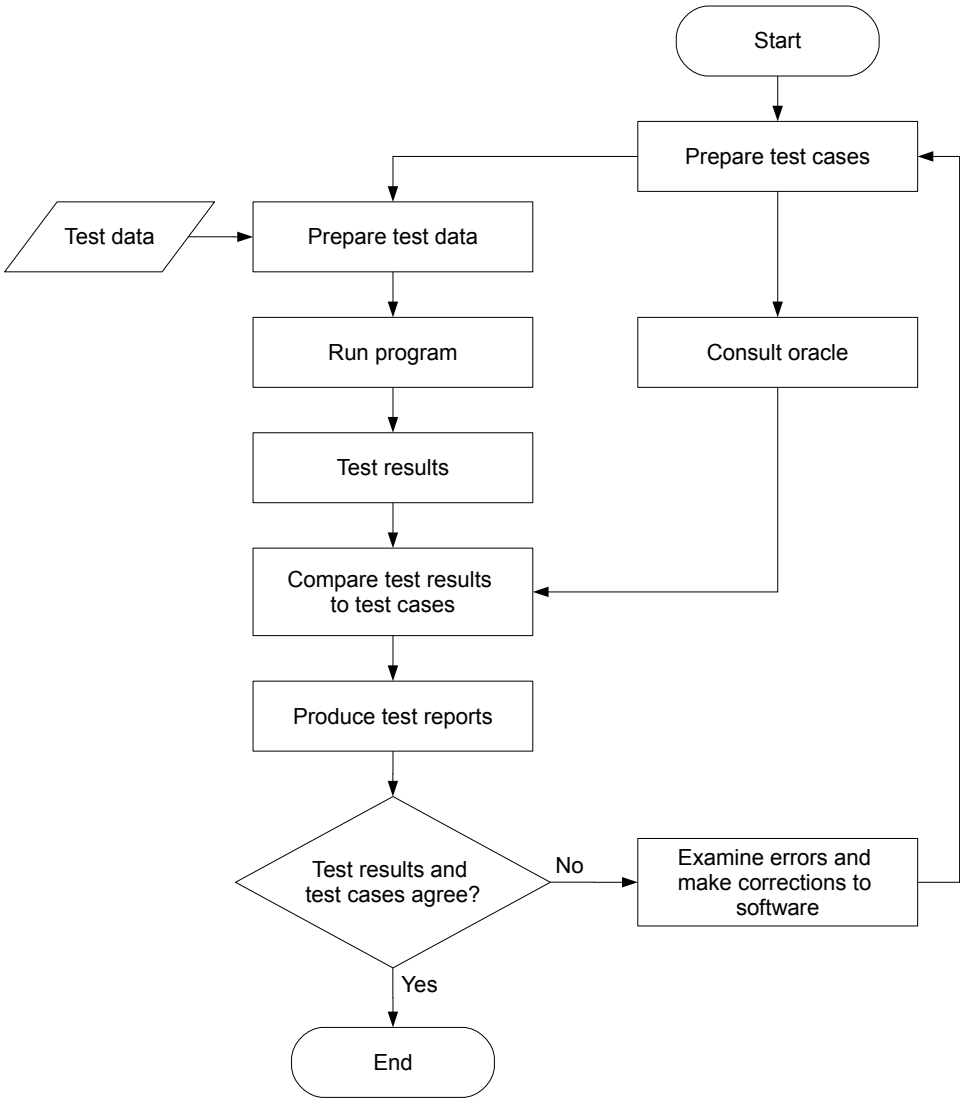


Figure 5.19: The software testing process. Components of FABCell were tested by comparing results to expected results and revising code if there were differences.

easier to verify where changes were affecting code.

5.11 Case studies

FABCell was validated by simulating a selection of models based on published models that results could be compared against. This is a similar to the testing strategies adopted by other toolkits such as AgentCell [38].

5.11.1 Game of Life

The classic example of a simple CA is Conway's Game of Life ([43]). Each cell shares a Game Of Life rule object and the migration state system is switched off so that cells cannot move. Every lattice site is occupied with a simple agent object maintaining a one-variable state: whether it is dead or alive. There are four rules determining whether a cell lives or dies based on the state of its neighbouring cells:

1. A live cell with fewer than two live neighbours dies.
2. A live cell with more than three live neighbours dies.
3. A live cell with two or three live neighbours is left unchanged.
4. A dead cell with exactly three live neighbours comes to life.

Each cell agent executes the four rules and then all cells are synchronously updated to reflect whether they are now dead or alive.

An example of a Game Of Life simulation is shown in Figure 5.20, which uses a Queen Bee Shuttle[9] as the starting seed. It demonstrates the algorithm in action and shows the evolution of the simulation to a stable point consisting of six still life blocks and two oscillators. This simulation is available as Movie `mov_gol_qbs` on the `CD-ROM` accompanying the thesis (see Appendix A).

[9]The Queen Bee Shuttle is a configuration where nine live cells are arranged in an arc formation which gives a well characterised pattern of behaviour in the automaton.

Figure 5.20: Simulation of The Game of Life cellular automata within FABCell. (a) Queen Bee Shuttle initial starting seed. nine cells are arranged in an arc formation which gives rise to a well characterised pattern of behaviour. (b) After 75 time steps. (c) After 150 time steps. (d) After 300 time steps the pattern is stable.

5.11.2 A cellular Potts model approach to morphogenesis

A fundamental aspect of developmental biology is morphogenesis and the study of tissue formation. Cells have the ability to differentiate and self organise into clusters. Work by [49] and [28] using CPM has shown one possible mechanism by which cells can do this. A cell sorting example [49] is used to illustrate how FABCell's CPM package can be used to study this phenomenon.

There are three types of entity in the model: light cells, dark cells and the medium. The latter acts as the background environment container and occupies all nodes on the lattice not occupied by light or dark cells. The following Hamiltonian was used to govern the system:

$$\begin{aligned} \mathcal{H} &= \sum_{(i,j),(i',j') \text{ neighbours}} J\left(\tau(\sigma(i,j)), \tau(\sigma(i',j'))\right) \\ &+ \lambda \sum_{a} [a(\sigma) - A_\sigma]^2 \end{aligned} \tag{5.11}$$

The first term describes the interaction energy between cell σ at node (i, j) and the cell at node (i', j') based on their type τ. The second term is a surface energy constraint that prevents the area of the cell $a(\sigma)$ from deviating significantly from its target area A_σ.

The equilibrium of an aggregation of cells was modelled starting from an initial random spread of light and dark cells (Figure 5.21) using the same parameters as Graner and Glazier [49]. By adjusting the interaction energies, cells can be made to aggregate into distinct groups based on their cell type. This simulation is available as Movie `mov_cell_sort` on the `CD-ROM` accompanying the thesis (see Appendix A).

5.12 Discussion

FABCell is a high performance agent based framework for simulating the dynamics of a system in a three-dimensional environment. The toolkit can be used to study problems using multiple model types: CA, LGCA, CPM and off-lattice models. Highly

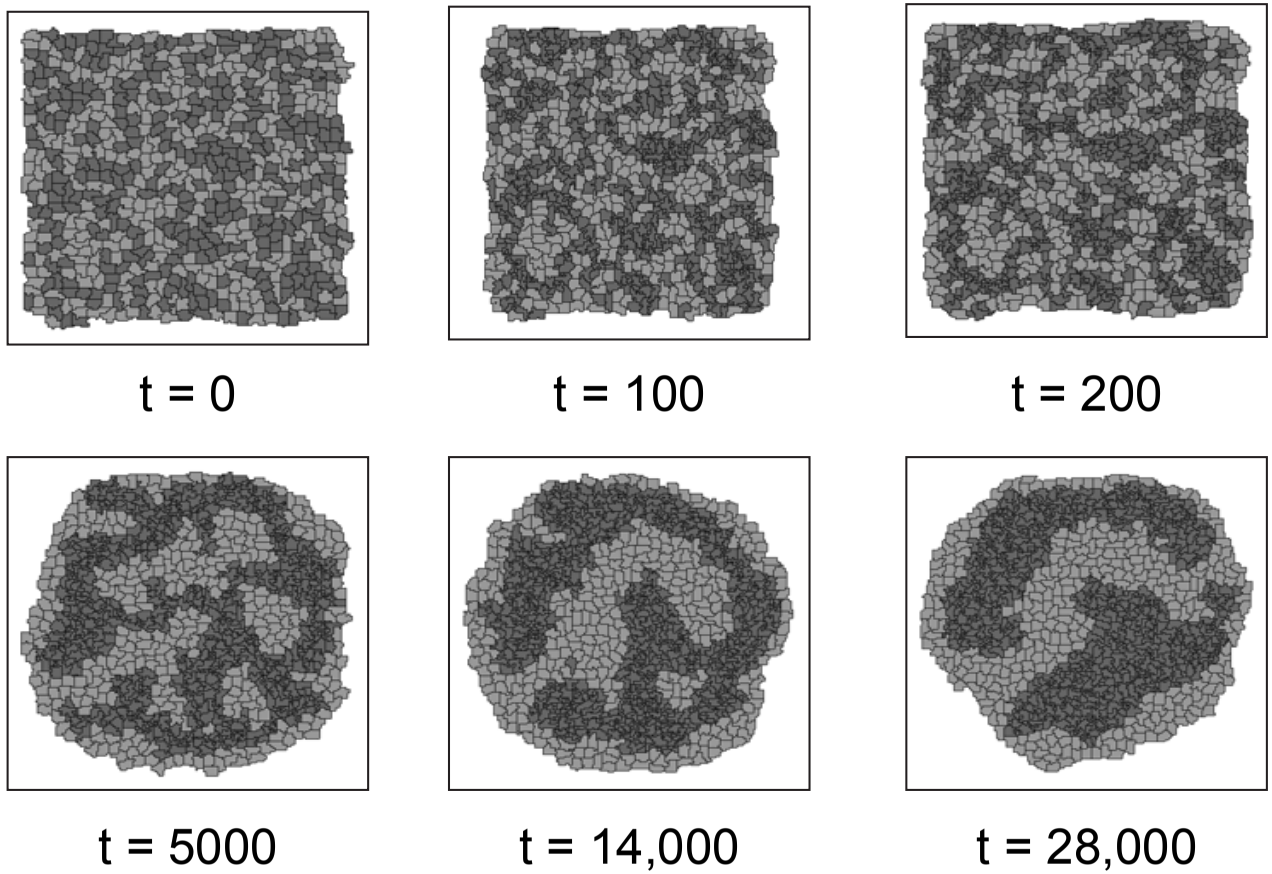


Figure 5.21: A simulation of cell sorting using a CPM with three different types of cell: light, dark and a medium cell representing the background environment. After 28000 time steps the light and dark cells have separated into two distinct clusters. For clarity the medium cell is not shown.

generic modelling tools, such as some of the tools mentioned in Section 5.3, often have a protracted development time and provide unnecessarily complex and inefficient solutions to agent based problems [197, 198]. Therefore, although FABCell has been designed as a framework for running different types of models, it is not a completely generic system and is biased towards the study of biological phenomena; performance is favoured over reusability.

FABCell was created using OO design patterns and is therefore highly structured and modular. This style of programming reduces redundancy in code and allows the system to be organised into a set of libraries, which are easier to maintain and update than one large monolithic program. It should be noted that an OO design will typically have a greater memory footprint than one written in a conventional procedural language. Therefore efficiency was a strong consideration during development. Wherever possible, the system maintains references to objects rather than copies to reduce the memory overhead. For example the rule system to govern a cell's behaviour allows cells to share rule objects since if several cells have the same behaviour, it is inefficient for each one to have a copy of the rule. The framework was developed using C++ because of its speed and portability and its ubiquitous use in scientific programming.

FABCell provides a highly adaptable environment for studying cell dynamics. Since no scale of representation is enforced, cells can be made of arbitrary size and complexity depending on the needs of the user. The environment could be considered to be one cell and used to model the internal movement of proteins or else host multiple cells as in the examples presented.

A number of common tasks can be automated and certain technical details can be further abstracted from users to reduce the amount of programming required to get a model running. A common methodology is allow users to use an XML schema to declare a set of directives specifying how a model should work which can subsequently be translated into a FABCell model.

The viewer was developed using the GLUT development library. This provides a windowing toolkit to allow OpenGL output to be rendered on screen. The viewer lacks standard windowed application features such as menus and tool bars. The controls are

largely keyboard driven requiring users to know a set of keyboard constants. This is still a learning barrier for users who prefer to use a mouse to control applications. It is envisaged that the next version of the viewer will switch to use a more modern window toolkit such as WxWidgets [122]. This provides support for OpenGL rendering and some preliminary work within the FABCell code base has already been started to migrate the code over to support a windowed environment.

The source code for FABCell is provided on the CD-ROM accompanying this thesis. Instructions for compiling and running the software and examples are given in Appendix F.

Chapter 6

A developmental model of FrzS translocation

Chapter 5 introduced FABCell, a general purpose biological modelling tool. In this and subsequent chapters, FABCell will be used to create mathematical and computational models to describe observable phenotypes in myxobacteria. The emphasis is on the models themselves with FABCell being used as a tool to explore and simulate ideas. For brevity, detailed technical information concerning the implementation of the models within FABCell will be succinct and kept to the minimum. Readers who are interested in the C++ implementation of FABCell can examine the FABCell source code base and resources on the `CD-ROM` accompanying this thesis.

6.1 Introduction

Myxobacteria are members of the δ-proteobacteria, distinguished by a complex and social life-cycle involving multicellular development [35, 37]. In response to starvation, cells pass through several developmental stages over a 72 h period, culminating in the formation of *fruiting bodies* within which dormant cell-types called myxospores form. The first of these stages is the *rippling* phase, occurring approximately four hours into starvation. The population self-organises into mobile bands of cells, which reflect off one another, giving the appearance of travelling waves. It is thought that rippling is an emergent property of an increased reversal frequency, brought about by a cell-cell contact mediated signalling pathway (C-signalling, transduced by the Frz pathway)

[84]. While much is known at the molecular level about components of the Frz signalling pathway [205], the way in which the pathway interacts with the machinery governing directional motion is poorly understood. Recent work has identified two polar proteins: FrzS and RomR, which dynamically relocate between poles during cell reversal [94, 110, 111]. Here, a model of the dynamic relocalisation of FrzS in response to Frz pathway activity is developed and implemented in an agent based simulation of myxobacteria rippling. Surprisingly such a model system describes many features of myxobacterial motility including organised ripple formation, oscillatory behaviour, fast and slow phases of protein relocalisation, a refractory period after reversal and robustness to variation in input signalling strength and duration. Here a novel model of dynamic protein relocalisation is presented, which comprises a self-regulating biochemical switch governing the transport of FrzS between cell poles. The model is sufficient to explain several observed features of myxobacteria motility and may be generally applicable to gliding bacterial locomotion.

6.2 Background

Myxobacteria are semi-flexible, rod-shaped cells that move in the direction of their long-axis, periodically reversing their direction of motion. The model myxobacterium, *Myxococcus xanthus* (*M. xanthus*) is approximately 5 μm to 7 μm long and 0.5 μm in diameter. During vegetative growth, cells move lengthwise reversing periodically, with an approximate frequency of 0.1 reversals/min [71]. When nutrients become scarce, cells initiate a programme of multicellular development culminating in the formation of *fruiting bodies*, large aggregates of approximately 100,000 cells. Approximately 10 % of cells entering the fruit differentiate into myxospores (dormant cell-types), which rest quiescent until nutrients become available once more. The developmental process involves a series of macroscopic changes in colony morphology. A key regulator of development is the C-signal, a cell surface-associated signal encoded by *csgA*, which is exchanged between cells in contact with one another. The expression of *csgA* is stimulated by C-signalling. Because of this positive feedback, C-signalling is thought to rise throughout development, and different colony morphologies are a consequence of

different C-signalling levels [86]. C-signalling is thought to affect colony morphology by altering the reversal frequency of individual cells in a contact-dependent fashion [84, 86, 89].

The first observable change in colony morphology during development is the emergence of rippling, which occurs during the first 4 h to 6 h of starvation. Cells synchronise their reversal cycles to form dense bands which travel as waves across the surface. Waves appear to pass through each other, however they are actually rebounding. It is thought that during a wave collision there is sufficient C-signalling between colliding cells to cause cells in opposing waves to reverse together [141].

Current understanding suggests that during rippling a C-signalling event is translated into a 'premature' reversal via the Frz signal transduction pathway, which is homologous to the chemosensory (Che) system in *Escherichia coli* [39, 103]. While many of the proteins of the Frz pathway have been characterised, the molecular basis for Frz signalling is poorly understood. Recent work has modelled the interactions of some of the key Frz proteins [63] from a network perspective, however recent evidence suggests one of the Frz proteins, FrzS is dynamically re-localised between cell poles. Thus the physical distribution of FrzS within a cell appears to be important in governing cell reversals [110].

Myxobacteria cells glide using two motility systems: the adventurous (A) motility system and the social (S) motility system [101] which at any one time are active at opposing poles. S-motility is coordinated at the leading pole; cells extend type IV pili which can adhere to the surface of other bacteria or polysaccharides, and upon retraction the cell is pulled forward. FrzS is a component of the S-motility engine. A-motility is less well understood. It is generally believed that cells extrude a slime from the lagging end which expands and generates a propulsive force to push cells forward [156, 196]. However, an alternative hypothesis has recently been proposed which suggests cells have focal adhesion systems that mediate gliding [112]. During a reversal, the leading pole of the cell becomes the lagging pole and vice versa, so the active S-motility and A-motility engines swap poles. It is unlikely that both motility systems are transported between cell poles, rather it is more plausible that both sets of

machinery exist at both poles and are switched on or off as required [109].

Rippling has previously been investigated using conventional CA models; however, lattice based models have an inherent number of limitations, restricting their accuracy when modelling spatially complex pattern formations. Alber et al. [2] present an LGCA rippling model which uses an internal variable speed clock to increment C-signal until it reaches a threshold, causing the system to reset and reverse direction. This clock model is sufficient to explain how collisions can synchronise cells to ripple; however, it hides much of the detail of the C-signalling mechanism. The C-signalling system is made up of a number of connected signalling pathways and although the clock model captures the macroscopic behaviour of the system as a whole, it cannot be used to investigate individual proteins and the networks themselves. The model presented here was developed as an alternative to the clock model so that there is an explicit representation of the pathways.

The previous *Frzilator* model [63] looked at the C-signalling and reversing; It uses a feedback loop involving FrzF, FrzCD and FrzE to explain periodic reversals; however, the feedback loop itself has not been shown to exist experimentally and there are still a number of unanswered questions concerning its sensitivity and response to signals. The parameter set given in Table 1 in [63] makes the *Frzilator* oscillate with a 10 min reversal cycle. It can be perturbed by C-signalling via increases in the active form of FrzF. The authors only discuss the behaviour of the system with short bursts of C-signal (0.5 min). Appendix C details a MATLAB® implementation of the *Frzilator* and the response of the system to stimuli. Figure C.1(b) shows the typical response of the *Frzilator* to signalling, which is characterised by a rapid increase in the FrzF, FrzCD and FrzE leading to a quicker reversal cycle in all three proteins. In a simulation of many cells, interactions can occur during every time step leading to relatively long bursts of signalling activity. Figure C.2 shows the effects of increasing the duration of signalling on the *Frzilator* from 10 min to 40 min whilst all other parameters are kept the same. Once $\Delta t > 10$ min there is a consistent, isolated reversal cycle at $t = 20$ whose profile does not match subsequent cycles; most notably, FrzF is affected during the refractory period. When $\Delta T \geq 40$ min, the system becomes chaotic. It is conceivable that cells will

experience signalling for a 40 min duration (there will be continuous signalling events even if the level of C-signalling may alter) making the *Frzilator* potentially unsuitable for such models.

The authors of the *Frzilator* also discuss its ability to cope with streaming (see Figure 3(c) in [64]). As the FrzF activation rate increases above 0.5 min^{-1}, the reversal frequency decreases. Figure C.3 shows the effect of a 20 min burst of signalling with an FrzF activation rate of 1.0 min^{-1} when it is started at various points during the reversal cycle. As with the results in Figure C.2, the system displays some inconsistent responses. When the start point is $t_0 = 8$ min, the system becomes unstable. This suggests that whilst the system is stable for very short bursts of signalling, it does not cope with long bursts of signalling particularly if the signalling starts and spans the duration of the refractory period.

The *Frzilator* offers an interesting model of the Frz pathway; however, its ability to deal with noisy bursts of signalling is unclear. The work in this chapter was motivated in part because the basis of the *Frzilator* has been called into question over the biology it proposes and in part to find a more robust and stable system for use in a large scale agent environment. A different approach to understanding how C-signal affects reversing is taken, which proposes a biochemical-feedback switch model of FrzS migration that displays oscillatory behaviour and can be modulated by input from a cell-cell contact dependent signal transduction pathway.

6.3 An oscillatory model of FrzS migration

The correlation between FrzS migration and cell reversal suggests that FrzS could be directly responsible for controlling cell movement, with input from the rest of the Frz pathway. To explore this idea a model of FrzS migration is developed.

The circuit shown in Figure 6.1 represents a model of the Frz signalling pathway. It incorporates a putative mechanism for the regulation of FrzS migration, and suggests how such a mechanism could respond to C-signalling. The scheme in Figure 6.1(b) will be referred to as the *Motilator*hereafter.

The Frz signal transduction pathway has been shown to play a key role in control-

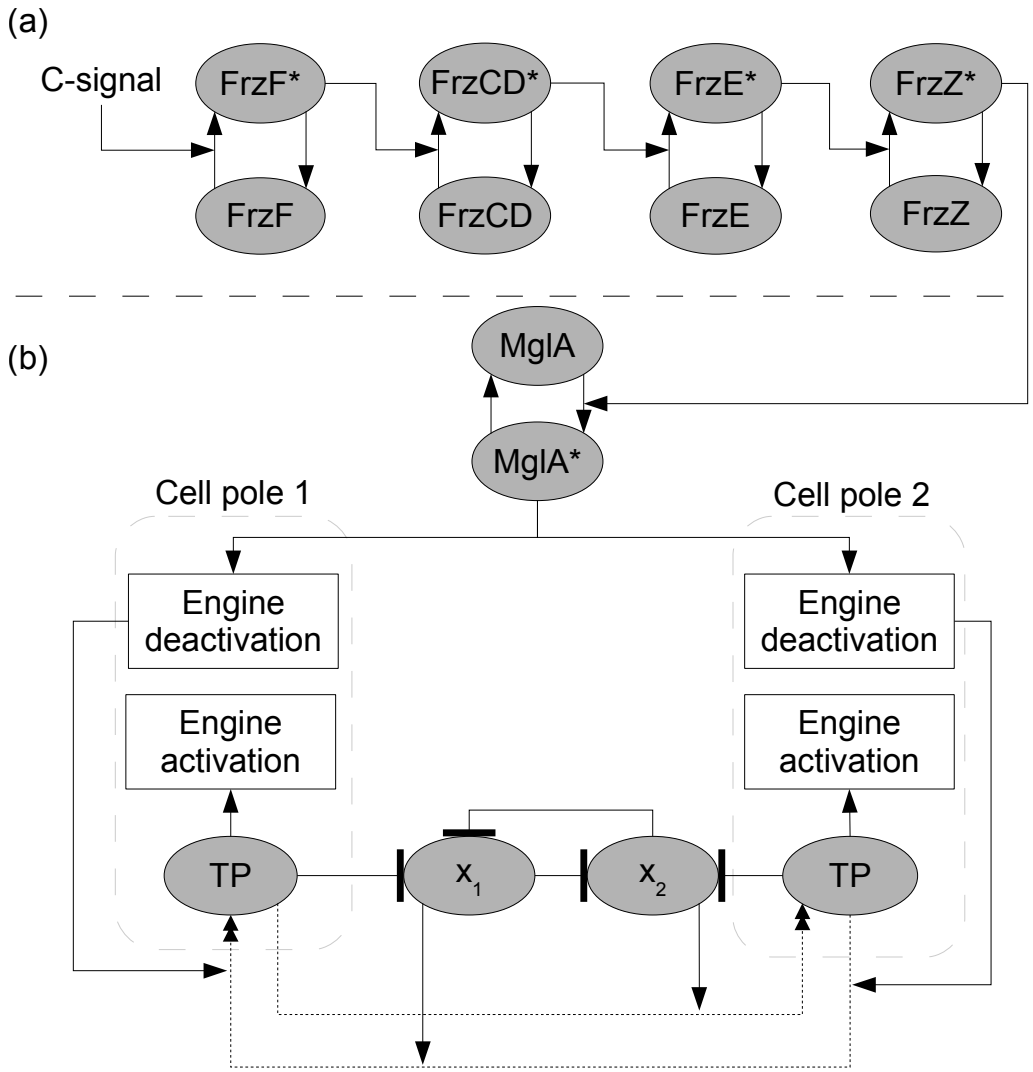


Figure 6.1: Schema of the Frz signalling and the *Motilator*. Dashed lines with double arrowheads represent protein transport mechanisms and ovals represent proteins. Where proteins have an inactive and active form, an asterisk (*) denotes the activated form of the protein. (a) The core Frz network which responds to C-signalling via FrzF. (b) The *Motilator,* a potential mechanism to explain FrzS protein translocation. A biological switch regulates the flow of protein between pole one and pole two (which can be considered the head and tail of a cell). x_1 controls an active transport mechanism from pole two to pole one and x_2 controls an active transport mechanism from pole one to pole two. Recruitment of FrzS at a pole is driven by engine deactivation. The build up of FrzS eventually stops recruitment at one pole whilst simultaneously it is recruited by the other pole.

ling cell motility [18, 204]; however, the molecular details of how all of the proteins interact have yet to be fully elucidated [205]. The core Frz pathway consists of five linked proteins: FrzF (a methyltransferase), FrzCD (a methyl-accepting chemotaxis protein, or MCP), FrzG (a methylesterase), FrzE (a hybrid histidine protein kinase) and FrzZ (a response regulator). Current understanding of Frz pathway activity suggests that C-signal causes activation of FrzF [102, 157] which methylates FrzCD. Methylated FrzCD triggers autophosphorylation of FrzE which in turn phosphorylates FrzZ [205]. FrzZ-P then regulates MglA activity, directly or indirectly, through an unknown mechanism. MglA plays a direct role in controlling cell reversal frequency [164] is required for the appropriate localisation of FrzS and RomR.

FrzS is a key component of S-motility and is thought to regulate the assembly and disassembly of Type IV pili at the cell poles [109, 110]. Recently it was shown that FrzS migrates between cell poles in a non-diffusive cyclic process which is synchronised with cell reversals [110]. During a reversal cycle, FrzS migrates from the lagging pole to the leading pole. A cell reversal occurs when the amount of FrzS is approximately equal at each pole [109]. FrzS continues to accumulate until an unknown switching mechanism causes it to start migrating back towards the lagging pole (which will subsequently become the leading pole). Simultaneously RomR accumulates in the lagging pole and is responsible for switching the A-motility machinery on and off [94]. RomR and FrzS function independently but their behaviour is somehow synchronised by MglA activity. Studies of the movement of green fluorescent protein (GFP)-tagged FrzS [110] and RomR-GFP [93] have shown that they move between cell poles in a manner synchronised with the cell reversal cycle. After accumulation at one cell pole the proteins begin to move back towards the other pole. Once the level of protein has dropped below a threshold, a rapid delocalisation occurs until all the protein has migrated to the other pole, whereupon the process repeats itself (see Figure 1 in [110]).

Mignot et al. [110] suggest that FrzS could be moved using a cytoskeletal track to account for its non-diffusing migration characteristics (see Figure 6.2). The *Motilator* proposes a plausible mechanism to explain how cells coordinate dynamic relocalisation of FrzS during cellular reversal. Cells are modelled as having a region at each pole

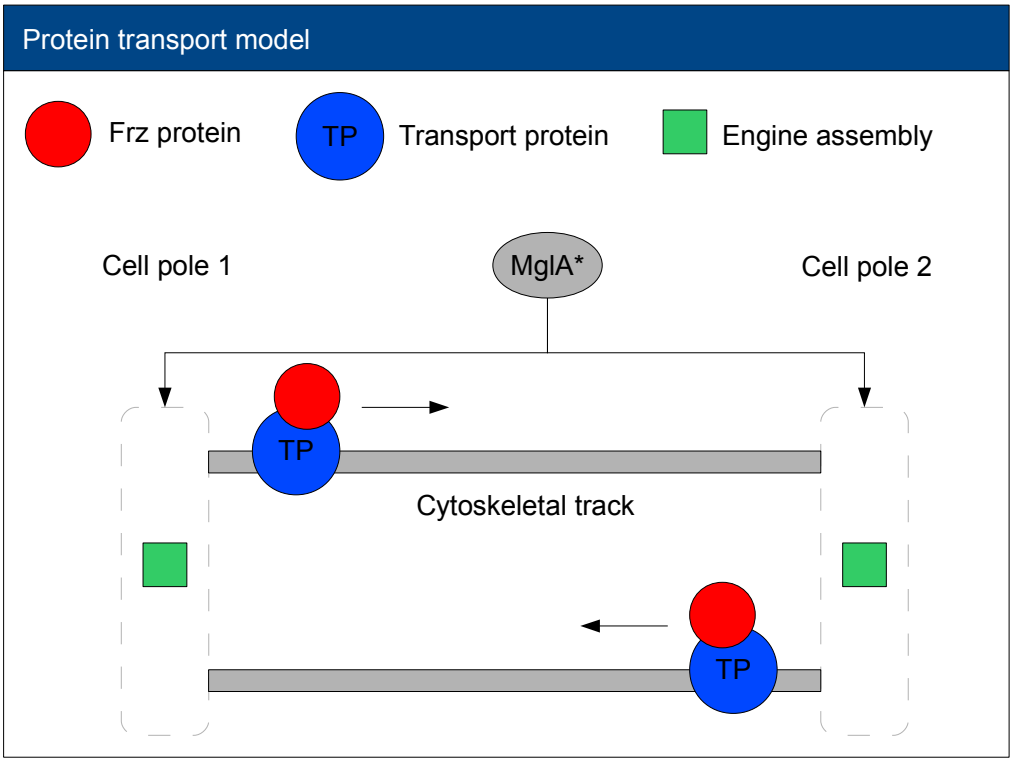


Figure 6.2: Theoretical protein transport model based on Mignot et al. (110). A transport complex (blue circles) moves Frz proteins (red circles) between cell poles (possibly along cytoskeletal tracks). The Frz proteins are required for engine assembly (green squares) and motility. The *Motilator* regulates the recruitment of the transport complex and how much of the Frz proteins are allowed to accumulate at a cell pole.

where FrzS can accumulate. FrzS can migrate between the two poles. An unspecified transporter protein complex (TP) controls migration between the poles. MglA triggers engine deactivation which triggers the recruitment of FrzS and the transporter protein. A bi-stable switch regulates the recruitment and is FrzS dependent. As FrzS accumulates, it eventually causes the switch to flip and turns off recruitment. Simultaneously at the other pole MglA is deactivating the engines so the FrzS is recruited to the other pole. MglA moderates the accumulation rate of FrzS at the poles, and so indirectly affects the switch; the faster FrzS accumulates, the faster the switch flips. In turn MglA is regulated by FrzZ, the output of the core Frz network (see Figure 6.1(a)).

6.4 The *Motilator* model of FrzS migration

The model of the dynamical properties of FrzS migration can be described by a set of ordinary differential equations (ODEs) and the behaviour of a cell population was simulated using an agent based model incorporating the ODE system.

The reaction scheme is divided into two parts: a model of the Frz pathway based upon current knowledge (see Figure 6.1(a)) and a model of how FrzS migration might potentially function (see Figure 6.1(b)). The output of the first model acts as an input to the second.

The first part of the scheme models the core Frz network comprising the interactions between FrzF, FrzCD, FrzE, FrzZ which is consistent with the biological data on the Frz pathway [205].

$$\frac{d}{dt}f = \frac{(1-f)k_f}{K_f + (1-f)} - \frac{fkd_d}{Kd_f + f} \tag{6.1}$$

$$\frac{d}{dt}c = \frac{(1-c)fk_c}{K_c + (1-c)} - \frac{ckd_c}{Kd_c + c} \tag{6.2}$$

$$\frac{d}{dt}e = \frac{(1-e)ck_e}{K_e + (1-e)} - \frac{ekd_e}{Kd_e + e} \tag{6.3}$$

$$\frac{d}{dt}z = \frac{(1-z)ek_z}{K_z + (1-z)} - \frac{zkd_z}{Kd_z + z} \tag{6.4}$$

where f is the fraction of activated FrzF, c the fraction of methylated FrzCD, e the fraction of phosphorylated FrzE and z the fraction of phosphorylated FrzZ. The equations are similar to the *Frzilator* model [63] although a feedback loop between FrzE and FrzF is not incorporated as this has not yet been show experimentally [169].

The second part of the problem is to consider the interaction between FrzZ and FrzS (mediated by MglA) and how FrzS might be transported through the cell. FrzS appears to move with two distinct phases: one slow and one fast. This prompted us to look at how biological switches [6, 56, 67, 88, 98] might be involved in regulation.

The core of the FrzS migration system (the *Motilator*), comprises two interlinked biological switches which regulate the flow of FrzS between two compartments: pole

1 at the head and pole 2 at the tail of the cell. In total the system comprises five main components: m, s_1, s_2, x_1 and x_2.

$$\frac{d}{dt}m = \frac{(1-m)zk_m}{K_m + (1-m)} - \frac{mk_{d_m}}{K_{d_m} + m} \tag{6.5}$$

$$\frac{d}{dt}s_1 = \frac{k_1 x_1 m(1-s_1)}{K_1 + (1-s_1)} - \frac{k_{d_1} x_2 m s_1}{K_{d_1} + s_1} \tag{6.6}$$

$$\frac{d}{dt}s_2 = \frac{k_2 x_2 m(1-s_2)}{K_2 + (1-s_2)} - \frac{k_{d_2} x_1 m s_2}{K_{d_2} + s_2} \tag{6.7}$$

$$\frac{d}{dt}x_1 = \frac{\alpha_1}{\beta_1 + x_2^p} - x_1\left(d_1 + s_1 k_1^{max}\right) \tag{6.8}$$

$$\frac{d}{dt}x_2 = \frac{\alpha_2}{\beta_2 + x_1^q} - x_2\left(d_2 + s_2 k_2^{max}\right) \tag{6.9}$$

where m is the fraction of MglA*, s_1 is the level of FrzS at pole 1 and s_2 the level of FrsS at pole 2, x_1 controls the migration from pole 2 to pole 1 and x_2 controls the migration from pole 2 to pole 1. The switch moderates the response of the system so that a large change in the level of FrzS is required to alter the behaviour of the system. The majority of FrzS must accumulate at one pole before the system switches to moving it to the other pole. This prevents protein from flowing back once it has started migrating. The *Motilator* is stimulated by the core Frz pathway (Eq. 6.1-6.4) via FrzZ acting on MglA which regulates FrzS migration. This subsequently affects how quickly x_1 and x_2 switch and therefore how quickly protein migration reverses. As FrzS accumulates at pole one it inhibits x_1, eventually triggering a switch once all the Frz has accumulated at the pole. This reverses migration causing FrzS to go to pole 2.

6.5 Model implementation

The ODE system was implemented in MATLAB® using the `ode45` function to solve it using numerical analysis. Stability analysis was also performed in MATLAB®. Simulations were carried out using FABCell (see Chapter 5) unless otherwise stated. The *Motilator* was reimplemented in C++ for the ODE solver in the GNU Scientific Library [41] so that it could used within FABCell. Simulations were carried out on a 2.53 GHz

Table 6.1: Parameter values for the *Motilator*.

Parameter	Value	Description
k_f, k_g, k_c, k_e, k_z	3.0 min^{-1}	
kd_f, kd_g, kd_c, kd_e, kd_z	1.0 min^{-1}	
K_f, K_g, K_c, K_e, K_z, Kd_f, Kd_g, Kd_c, Kd_e, Kd_z	1.0	Dimensionless Michaelis-Menten constant.
a_1	3.0	Governs the production of x_1.
a_2	3.0	Governs the production of protein x_2.
β_1	1.0	Dimensionless.
β_2	1.0	Dimensionless.
p	3.0	Co-operativity between x_1 and x_2.
q	3.0	Co-operativity between x_2 and x_1.
k_m	3.0 min^{-1}	
K_{d_m},K_m,K_m	1	Dimensionless Michaelis-Menten constant.
k_1,k_{d_1},k_2,k_{d_2}	1.0 min^{-1}	
K_1,K_{d_1},K_2,K_{d_2}	0.005	Dimensionless Michaelis-Menten constant.
d_1	1.0 min^{-1}	Minimum degradation constant for x_1.
d_2	1.0 min^{-1}	Minimum degradation constant for x_2.
k_1^{max}	0.8 min^{-1}	
k_2^{max}	0.8 min^{-1}	

Intel® Core™2 workstation with 4 GB of RAM. FABCell was compiled using GCC 4.0 [42] with the OpenMP extensions on the 32 bit version of the Ubuntu™ 9.04 [95] GNU/Linux distribution. Simulations were suitable for execution on supercomputers, but were restricted to multiple core workstations; simulations typically generate several gigabytes of data and local high performance computing resources did not allow such large amounts of data to be stored on their systems. Remote transfer of gigabytes of data across a network was also not practical given the number of simulations performed.

6.6 Stability analysis

In reality, a biological system such as the *Motilator* will be responding consistently and predictably to noisy stimuli and signals, so the stability of the *Motilator* is an important issue. The *Motilator* should be periodic, and should remain so even when challenged with noisy stimuli. To assess the stability of *Motilator* it was analysed for stable limit cycles. The system should oscillate about a mean and return back to this stable

condition after being perturbed.

Since the *Motilator* is non-linear, an explicit solution cannot be obtained. Instead the equilibria of Equations 6.5 to 6.9 were analysed to assess stability [25, 171]. Analysis is based on the study of an individual *Motilator* system in MATLAB®.

The variables m, s_1, s_2, x_1 and x_2 represent proteins concentrations. The steady state solutions are when the following conditions hold:

$$\frac{(1-m)zk_m}{K_m+(1-m)} - \frac{mk_{d_m}}{K_{d_m}+m} = 0 \tag{6.10}$$

$$\frac{k_1x_1m(1-s_1)}{K_1+(1-s_1)} - \frac{k_{d_1}x_2ms_1}{K_{d_1}+s_1} = 0 \tag{6.11}$$

$$\frac{k_2x_2m(1-s_2)}{K_2+(1-s_2)} - \frac{k_{d_2}x_1ms_2}{K_{d_2}+s_2} = 0 \tag{6.12}$$

$$\frac{\alpha_1}{\beta_1+x_2^p} - x_1\left(d_1+s_1k_1^{max}\right) = 0 \tag{6.13}$$

$$\frac{\alpha_2}{\beta_2+x_1^q} - x_2\left(d_2+s_2k_2^{max}\right) = 0 \tag{6.14}$$

The fraction of MglA* (m) is independent of the other quantities and can be solved explicitly using the parameters in Table 6.1.

$$\frac{(1-m)zk_m}{K_m+(1-m)} - \frac{mk_{d_m}}{K_{d_m}+m} = \frac{m}{0.05+m} - \frac{1-m}{1.05-m} = 0 \tag{6.15}$$

$$m = 0.5 \tag{6.16}$$

Since m is solvable and only takes one value, s_1, s_2, x_1 and x_2 can be solved. The nullclines for s_1 and s_2 can be expressed, respectively, as

$$\frac{k_1x_1m(1-s_1)(K_{d_1}+s_1)}{k_{d_1}ms_1(K_{s_1}+(1-s_1))} - x_2 = 0 \tag{6.17}$$

$$\frac{k_{d_2}x_1ms_2(K_2+(1-s_2))}{k_{d_2}m(1-s_2)(K_{d_2}+s_2)} - x_2 = 0 \tag{6.18}$$

The nullclines for x_1 and x_2 can be expressed, respectively, as

$$\frac{\alpha_1}{(\beta_1 + x_2^p)(d_1 + s_1 k_1^{max})} - x_1 = 0 \tag{6.19}$$

$$\sqrt[q]{\frac{\alpha_2}{x_2(d_2 + s_2 k_2^{max})} - \beta_2} - x_1 = 0 \tag{6.20}$$

The equilibria solutions for s_1 and s_2, x_1 and x_2 must be obtained simultaneously due to the interdependence of the species. There are multiple solutions both real and complex. Table 6.2 lists only the real solutions since we are dealing with a biological system. Solutions p_3, p_4 and p_6 contain negative numbers so they are not considered further since a biological species can be neither non-negative nor complex; it either exists or it does not.

Table 6.2: Solution sets for the intersections of the nullclines of Equations 6.5 to 6.9. Only real solutions are given.

	m	s_1	s_2	x_1	x_2
p_1	0.5	0.989	1.062	1.629	0.305
p_2	0.5	1.062	0.989	0.305	1.629
p_3	0.5	-0.051	1.051	3.127	0.052
p_4	0.5	1.051	-0.051	0.052	3.127
p_5	0.5	0.5	0.5	1.0277	1.0277
p_6	0.5	0.5	0.5	-1.369	-1.369

A quantifiable measure of the system's stability can be found by considering the Eigenvalues of the Jacobian matrix of Equations 6.5 to 6.9 (see Appendix B) and determining the roots of the characteristic equation.

Within the *Motilator* model each cell has a duplicate machinery at each pole giving rise to symmetric parameters. Solution sets p_1 and p_2 are symmetric and therefore share the same Eigenvalues:

$$\lambda_1^{1,2} = -51.3076$$
$$\lambda_2^{1,2} = -10.9992$$
$$\lambda_3^{1,2} = -2.61113$$
$$\lambda_4^{1,2} = -1.04895$$
$$\lambda_5^{1,2} = -0.330579$$

These solutions sets are both stable nodes.

Solution set p_5 is unstable since the real part of the two complex roots are positive:

$$\lambda_1^5 = -3.585$$
$$\lambda_2^5 = 0.307 + 0.720i$$
$$\lambda_3^5 = 0.307 - 0.720i$$
$$\lambda_4^5 = -0.331$$
$$\lambda_5^5 = -0.170$$

The solutions sets can tell us how stability affects the periodicity of the system. For these experiments, s_1 was studied as it directly affects cell reversal. Figure 6.3 shows how varying the initial condition for s_1 around the equilibrium p_1 affects the solution of s_1 over time. Figure 6.4 shows how varying the initial start condition x_1 around the equilibrium p_1 affects s_1. In both cases, there is no periodic solution indicating that as a stable point, solutions always tend towards the equilibrium and cannot oscillate.

Figure 6.5 and Figure 6.6 show respectively how varying the initial conditions for s_1 and x_1 around the equilibrium p_5 affects s_1. In both case (and also for s_2 and x_2) the solution remains periodic. Solutions around the unstable equilibrium indicate that not only is the system periodic but it is robust and resistant to noise since the periodicity remains stable even with a large variation of the parameters.

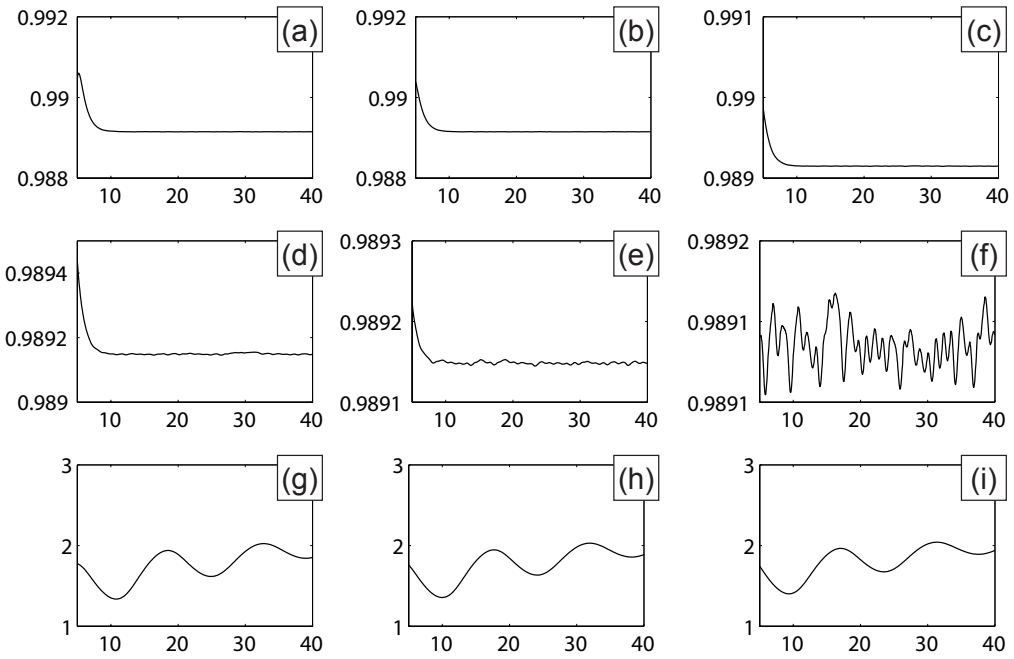


Figure 6.3: The effect of varying s_1 for solution p_1 where $m = 0.5$, $s_2 = 1.062$, $x_1 = 1.629$ and $x_2 = 0.305$. There is a loss of periodicity around the equilibrium. (a) $s_1 = 0$. (b) $s_1 = 0.2$. (c) $s_1 = 0.4$. (d) $s_1 = 0.6$. (e) $s_1 = 0.8$. (f) $s_1 = 1.0$. (g) $s_1 = 1.2$. (h) $s_1 = 1.4$. (i) $s_1 = 1.6$.

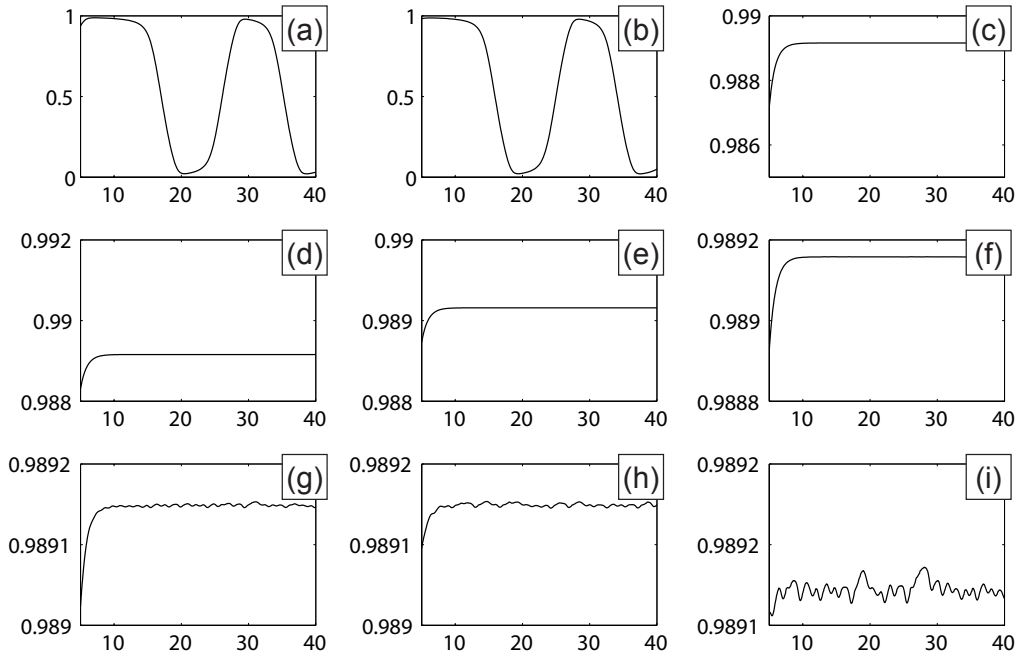


Figure 6.4: The effect of varying x_1 for solution p_1 where $m = 0.5$, $s_1 = 0.989$, $s_2 = 1.062$ and $x_2 = 0.305$. There is a loss of periodicity around the equilibrium. (a) $x_1 = 0$. (b) $x_1 = 0.2$. (c) $x_1 = 0.4$. (d) $x_1 = 0.6$. (e) $x_1 = 0.8$. (f) $x_1 = 1.0$. (g) $x_1 = 1.2$. (h) $x_1 = 1.4$.(i) $x_1 = 1.6$.

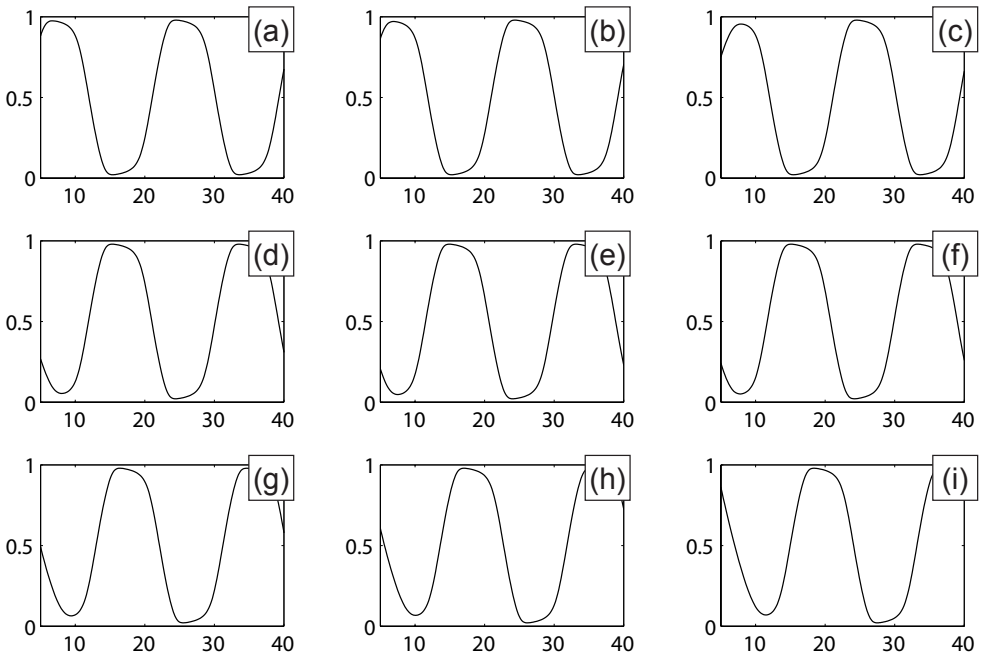


Figure 6.5: The effect of varying s_1 for solution p_5 where $m = 0.5$, $s_2 = 0.5$, $x_1 = 1.0277$ and $x_2 = 1.0277$. The system remains stable and periodic under all conditions. (a) $s_1 = 0$. (b) $s_1 = 0.2$. (c) $s_1 = 0.4$. (d) $s_1 = 0.6$. (e) $s_1 = 0.8$. (f) $s_1 = 1.0$. (g) $s_1 = 1.2$. (h) $s_1 = 1.4$. (i) $s_1 = 1.6$.

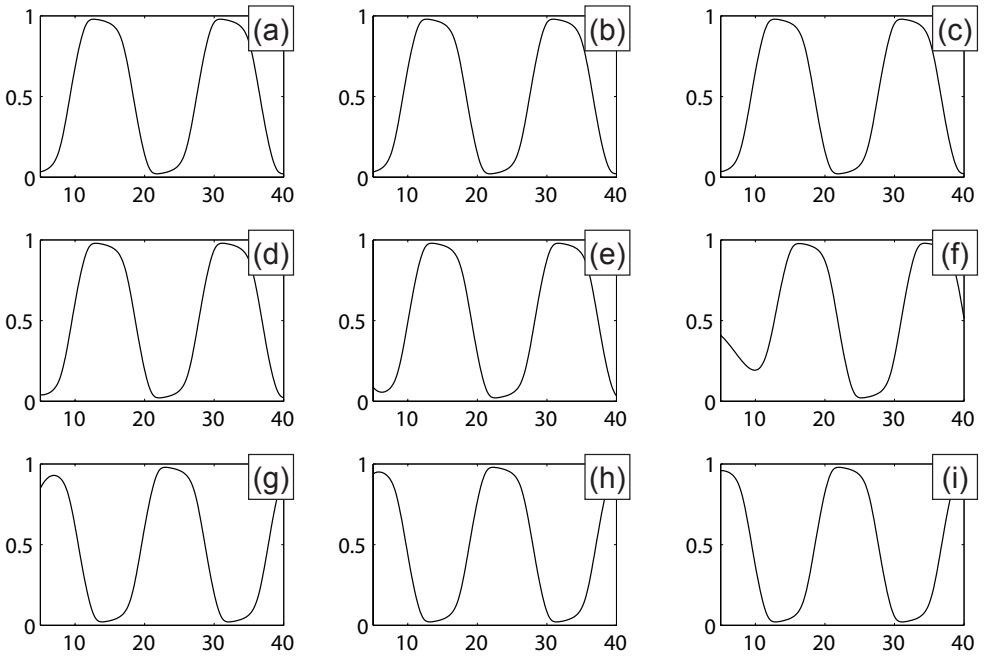


Figure 6.6: The effect of varying x_1 for solution p_5 where $m = 0.5$, $s_1 = 0.5$, $s_2 = 0.5$ and $x_2 = 1.0277$. The system remains stable and periodic under all conditions. (a) $x_1 = 0$. (b) $x_1 = 0.2$. (c) $x_1 = 0.4$. (d) $x_1 = 0.6$. (e) $x_1 = 0.8$. (f) $x_1 = 1.0$. (g) $x_1 = 1.2$. (h) $x_1 = 1.4$.(i) $x_1 = 1.6$.

6.7 Results

6.7.1 Signalling and reversing

The core Frz circuit (see Figure 6.1(a)) is a linear cascade without feedback. Each of the proteins is inactivated over time so that a continuous level of C-signalling (via FrzF) is required to maintain output (FrzZ*) to the *Motilator*. This in turn maintains activation of MglA which controls the FrzS translocation system. In response to MglA activation (corresponding to an increase in C-signalling), reversal period decreases (see Figure 6.7). Activated MglA increases the rate of flow of FrzS between cell poles, increasing the frequency of cell reversals. Experimental results have shown that FrzF mutants do not reverse [12, 169] so the model assumes there is always a small residual amount of C-signal within the cell to maintain an oscillation period. In the complete absence of C-signalling (an FrzF mutant) there are no oscillations and the cell does not reverse.

By tuning the *Motilator* to reverse every ten minutes, the system exhibits a minimum reversal period of three minutes which agrees with evidence showing that cells at high cell density reverse after three or four collisions with other cells (corresponding to about three or four minutes of activity) [154].

The *Motilator* is only sensitive to signalling at certain times. Immediately after the FrzS has migrated to one pole there is a 3-4 minute phase during which the system is insensitive to MglA* (see Figure 6.7(b)). This can serve as a natural refractory period which is thought to be a feature of myxobacterial rippling [21, 62].

6.7.2 Oscillations

Reversal data for cells indicates cells do not stop reversing completely but simply have a very low reversal frequency so the *Motilator* was designed to have a continuous oscillatory response to C-signalling. Figure 6.8 shows the relationship between MglA* and reversal frequency. As MglA* decreases there is a corresponding decrease in reversal frequency allowing us to have cells with a very low reversal rate at very low levels of C-signal.

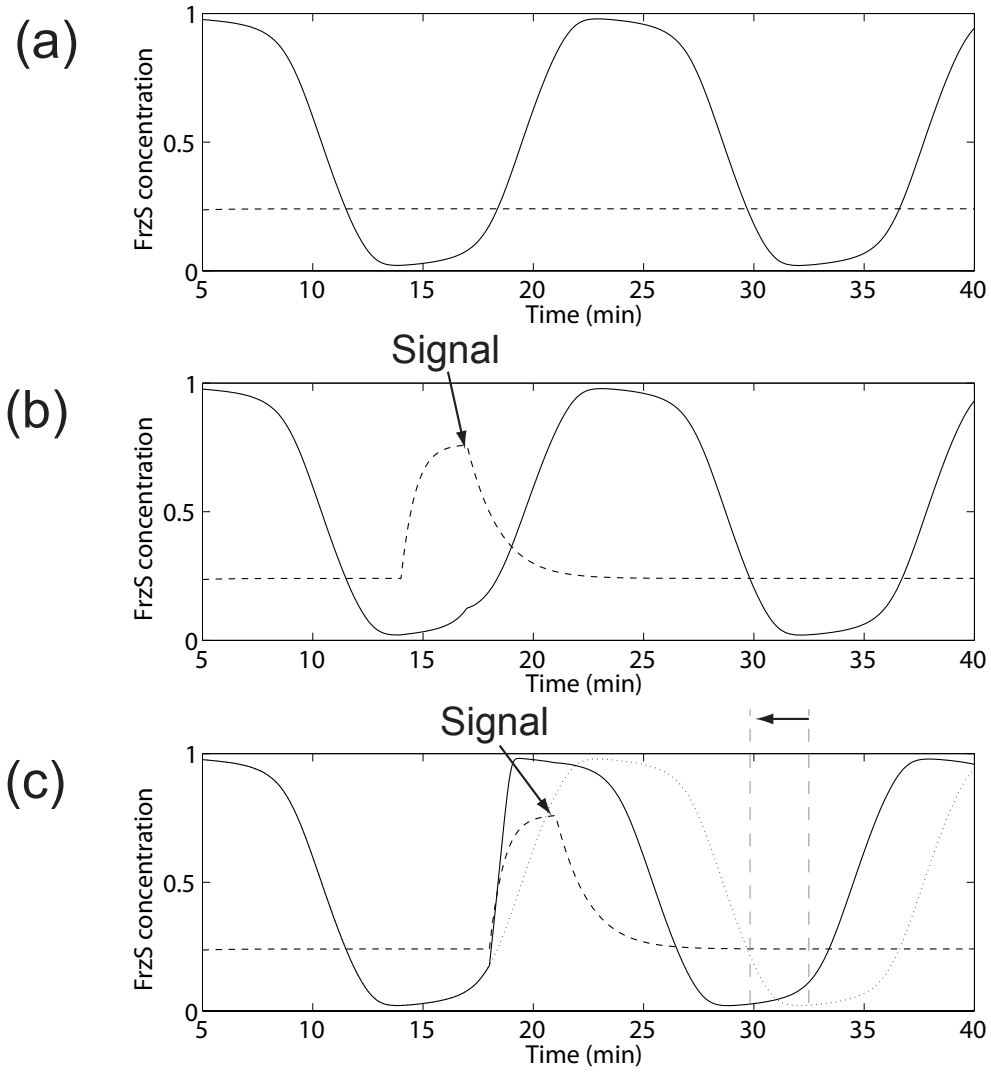


Figure 6.7: Refractory period characteristics of the *Motilator* with initial conditions $x_1 = 2$, $x_2 = 0$, $s_1 = 1$ and $s_2 = 1$. The concentration of FrzS at one pole and the level of MglA* are shown by solid lines and dashed lines respectively. (a) The system has been tuned to oscillate with a period of ten minutes in response to a low level of MglA*. (b) A burst increase in MglA* between $t = 14$ and $t = 17$ does not perturb the oscillation corresponding to a refractory period just after all of the FrzS has migrated to one pole. (c) A burst increase in MglA* between $t = 18$ and $t = 21$ speeds up the reversal cycle (the dotted line shows the position of the unperturbed system) corresponding to the end of the refractory period.

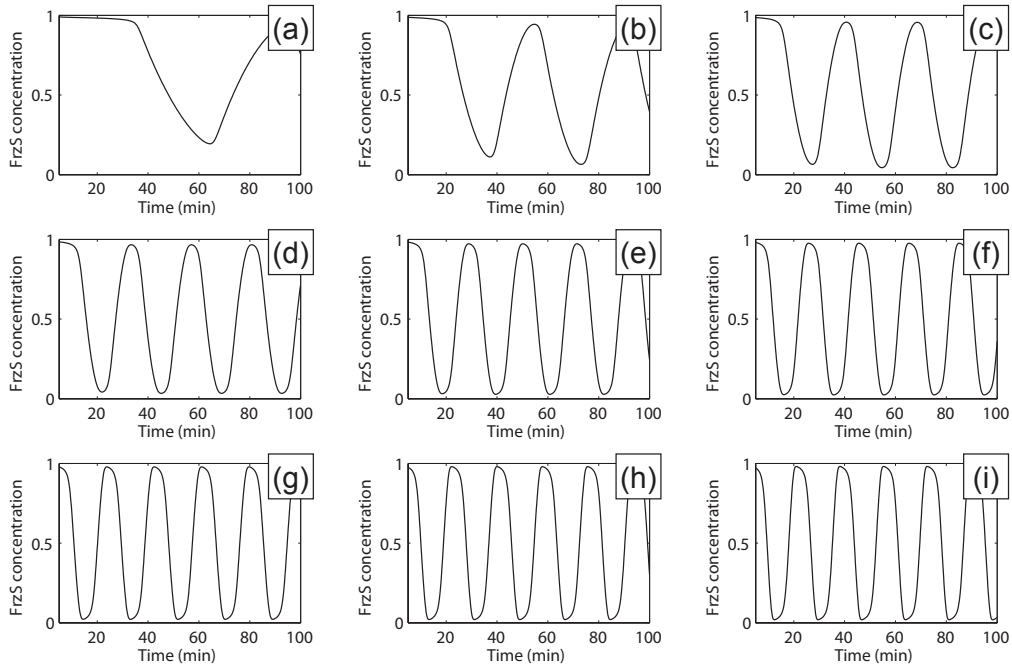


Figure 6.8: The response of the *Motilator* to different levels of MglA* (m) signalling. (a) $m = 0.02$. (b) $m = 0.04$. (c) $m = 0.06$. (d) $m = 0.08$. (e) $m = 0.1$. (f) $m = 0.12$. (g) $m = 0.14$. (h) $m = 0.16$. (i) $m = 0.18$.

The *Motilator* was tuned to have a reversal period of ten minutes corresponding to the reversal period in isolated myxobacteria [71]. In the agent based simulation, cell interactions are random, therefore C-signalling is noisy; however, the system still displays stable and robust oscillations.

Data from Mignot et al. [110] and Leonardy et al. [93] shows that GFP tagged FrzS migrates between cell poles in a well-defined manner, with migration occurring in two phases: one slow and one fast. During the reversal cycle, FrzS migrates slowly for the first 5 min, followed by a switch to a much more rapid phase of migration when the bulk of the protein rapidly disassociates from one pole and moves to the other pole. Figure 6.9 shows a comparison between GFP tagged FrzS [110] and the response of the *Motilator*. The switch exhibits a slight lag during its response to signalling, hence migration is slow at the start until FrzS has accumulated to a sufficient level to cause the switch to flip, after which FrzS migrates quickly out of the compartment.

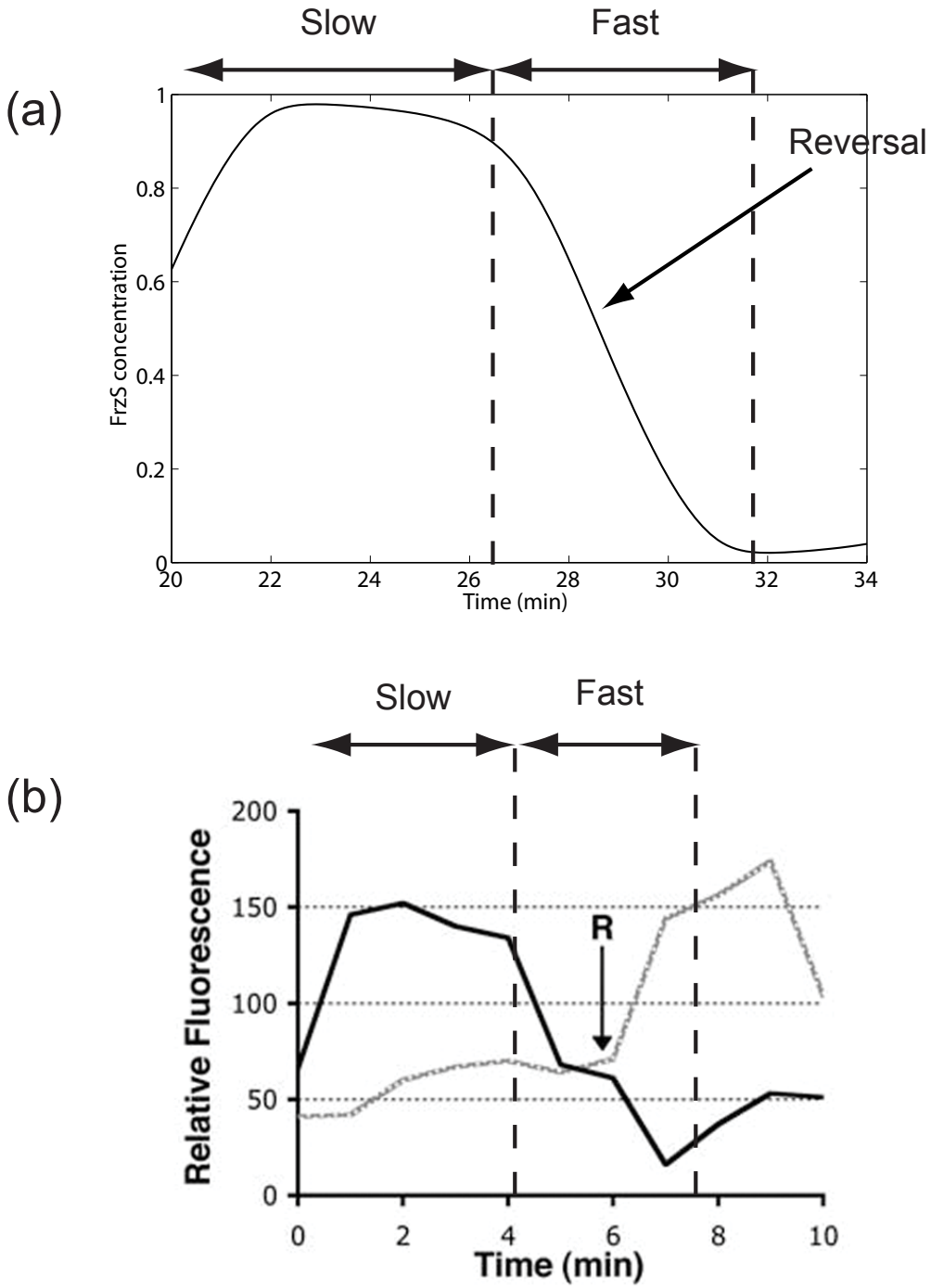


Figure 6.9: Response of the *Motilator* showing FrzS movement during a ripple cycle. The system exhibit two migration phases: slow and fast. FrzS initially migrates slowly for the first 5 min followed by sudden increase until all but a small residual amount of FrzS has moved to one cell pole, matching experimental data (110). A cell reversal occurs during the fast phase. (a) Response of the *Motilator*. (b) Corresponding movement of GFP tagged FrzS protein with a cell (data and image adapted from (110)).

6.7.3 Agent based simulation

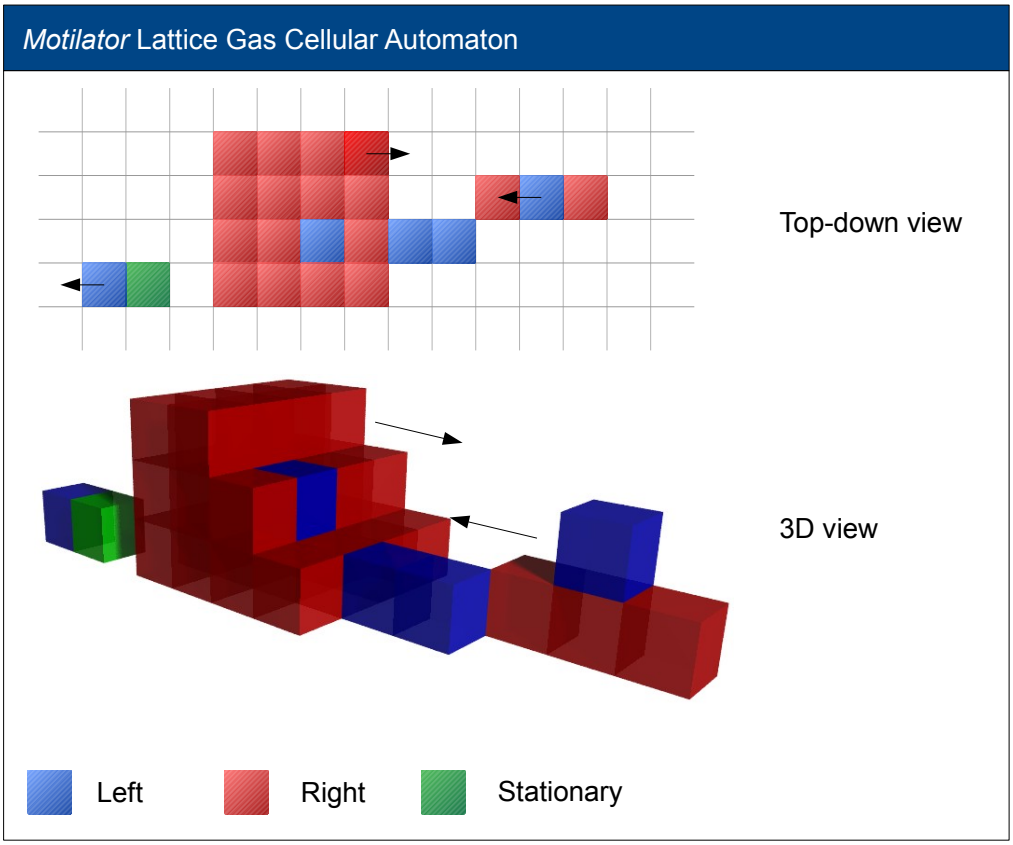


Figure 6.10: LGCA model for simulating the *Motilator* in a cell population. The LGCA features three movement channels cells can move in: left (blue nodes), right (red nodes) and stationary (green nodes). Movement channels restrict cell motion and approximate the generally straight line motion of myxobacteria cells. Cells can stack on top of each other at lattice locations.

To validate the *Motilator* model a cellular automata (CA) agent-based simulation comprising 45,000 *M. xanthus* cells was created using FABCell (see Figure 6.10). Each cell was governed by the *Motilator*. An agent based model was chosen over a differential equation model because of the nature of the model. Partial Differential Equation (PDEs) models have been used to study rippling [62]; however, they do not adequately capture the stochastic nature of an interacting population of cells nor the spatial effect of cell interactions (see Chapter 3). Signalling for example is not a continuous phenomenon; it occurs only when two cells happen to be close enough to signal. Each cell will behave slightly differently depending on its internal state.

Cells were randomly distributed onto a $300 \times 50 \times 30$ three-dimensional lattice so

that the average density at a given location in the xy-plane is 3. Assuming an average cell length of five μm, the simulated volume has dimensions 1500 μm $\times$ 250 μm $\times$ 150 μm. The model is designed similarly to that used by Börner et al. [20, 21]. Each cell has an orientation $\theta \in \{0, \pi\}$ describing its orientation about the z-axis and specifying the direction of travel. Cells move continuously but are restricted to moving in predefined rows to simulate the effects of alignment from slime and S-motility. They rest with a small probability $p \leq 0.05$. During a simulation iteration, each cell will check a local neighbourhood of nodes ahead of it for any cells in close proximity. Any neighbourhood cells with an orientation different to the current cell ($\Delta\theta > 0$) are considered to be colliding with the cell and hence trigger C-signalling and any other contact dependent signals which affects the *Motilator*. If there are sufficient collisions to trigger a cell reversal, the cell will remain stationary for one time step reflecting the time it takes for the motility machinery to switch between cell poles so the cell can go in the opposite direction. After a reversal, a cell will enter a refractory phase for τ time steps during which time it will not respond to C-signal but it can stimulate other cells. If a cell encounters an oncoming wave of cells, it will push into the wave, moving either above or below the blocking cell that is directly in front of it with a probability $p = 0.5$ and displacing the cells above it.

Ripple formation is characterised by cells grouping into travelling wave clusters. A proximity function $P(x)$ (similar to the correlation function presented by Wu et al. [202]) measures how close cells get to each other over time.

$$P(x) = \frac{1}{N(x)} \sum_{\langle i,j \rangle, i \neq j}^{C(x)} \frac{1}{\left(|\boldsymbol{l_i} - \boldsymbol{l_j}|\right)^2}, \tag{6.21}$$

where $\boldsymbol{l_i}$ is a coordinate vector specifying the location of cell i, $\boldsymbol{l_j}$ is a coordinate vector specifying the location of cell j, $C(x)$ is a list of cells a distance x from each other and $N(x)$ is the total number of cells a distance x from each other.

Figure 6.11 shows how the proximity of cells changed over time during three independent trials. There is an increase in proximity corresponding to cells getting closer (the response is inversely proportional so the close cells become, the higher the

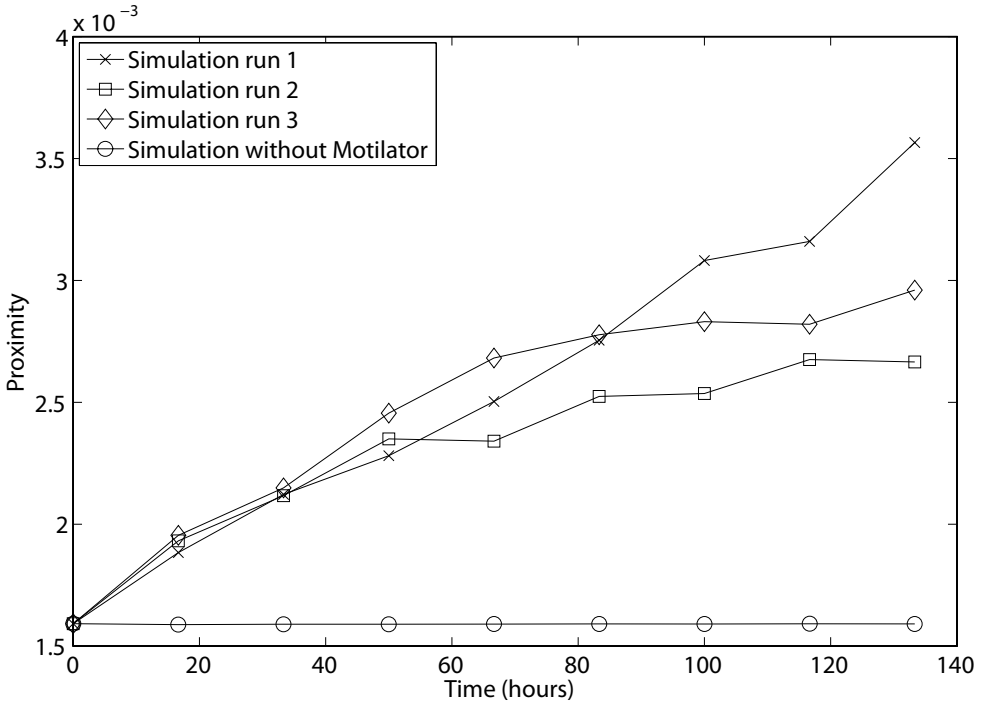


Figure 6.11: Variation of the proximity function for three independent simulation runs and a control simulation where the *Motilator* was switched off. Without the *Motilator*, cells move independently of each other and remain randomly distributed so their proximity does not improve.

proximity function). As t increases the rate of change of P decreases as cells become organised into ripples and the distance between then stabilises.

$$\lim_{t\to\infty} \frac{dP}{dt} = 0 \tag{6.22}$$

A simulation run where the *Motilator* was switched off was also performed. Without the *Motilator*, cells are not affected by collisions and behave largely independently of each other. They remain essentially randomly distributed so their average distance from each other does not change and their proximity measure does not improve. This simulation is available as Movie `mov_mot_no_coll` on the `CD-ROM` accompanying the thesis (see Appendix A).

Figure 6.12 shows the space time plots of a simulation looking at the space-time dynamics of the x axis run at the beginning, middle and end of a simulation run illustrating the formation of ripples. The space-time plot measures how the x position of each cell changes over time. The characteristic saw tooth oscillation pattern corres-

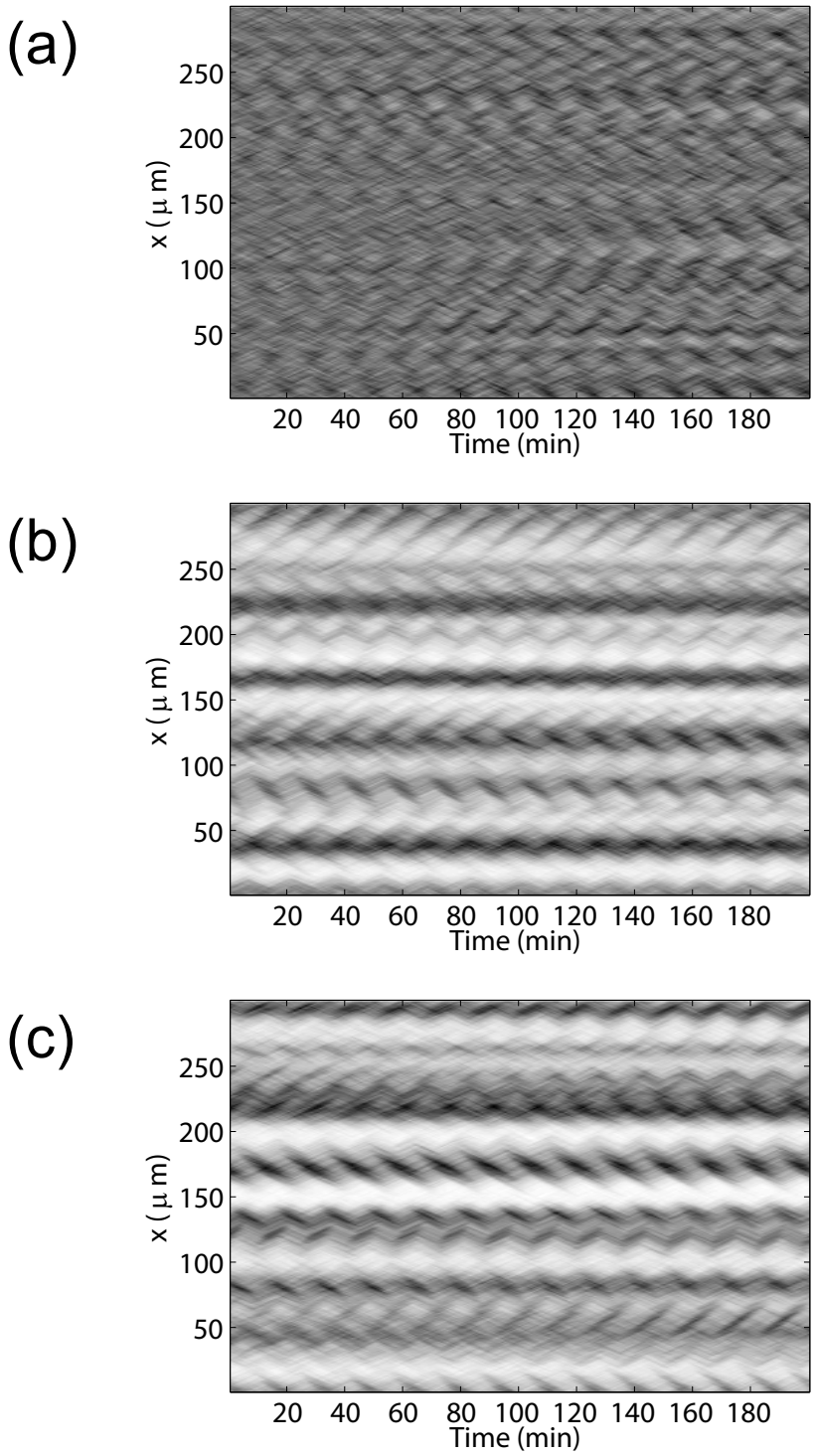


Figure 6.12: Space time plot showing the evolution of an agent based model controlled by the *Motilator*. The movements of 45,000 cell were recorded along the x-axis for 200 min intervals. From an initial random distribution, cells coordinate into five distinct ripple bands. Once a cell has joined a ripple it stays in that ripple with a very high probability. (a) 100 min. (b) 2000 min. (c) 4000 min.

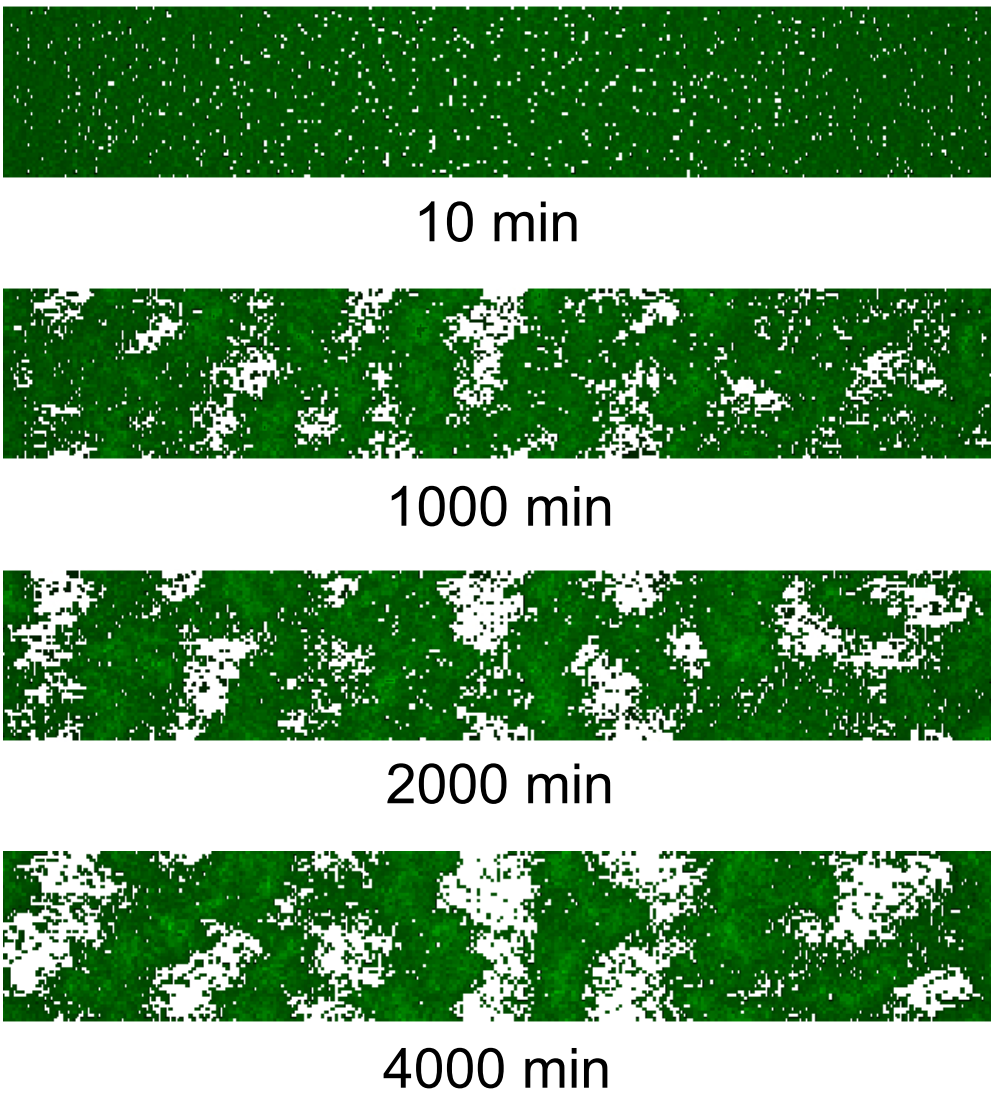


Figure 6.13: Density plots of cell movement in the xy-plane for a *Motilator* simulation of 45,000 cells. Each cell uses the *Motilator* to control its behaviour. Cells coordinate into ordered ripple bands which become more defined with time and eventually all individual cells join a ripple.

ponds to continual reversals. The darker oscillations show that three major and two minor ripples form which remain stable. Once the cells are in a ripple, they leave it with only a small probability. The *Motilator* causes cells to isolate into very distinct ripples which are often separated from each other by several wavelengths. The average reversal period of a cell in a ripple is 11 min. Nearly all of the cells eventually join one of the ripples causing a very low cell density between ripples. Figure 6.13 shows a snapshot of a simulation showing the formation of ripples over a period of 4000 min. The cells coordinate their actions into five major ripples corresponding to the wave fronts in Figure 6.12. This simulation is available as Movie `mov_mot` on the `CD-ROM` accompanying the thesis (see Appendix A).

6.8 Discussion

The lack of experimental evidence into how the Frz pathway interacts with FrzS and RomR prompted us to develop a biologically plausible model to explain protein transport. Leonardy et al. [93] and Mignot et al. [110] show that the migration patterns of FrzS and RomR are synchronised with cell reversal, and they are intrinsic to the activation of the adventurous and social motility machinery. Igoshin et al. [63] showed that the introduction of a negative feedback loop into the Frz pathway (the *Frzilator*) causes the levels of key Frz proteins to oscillate, with the periodicity regulated by FrzF.

The presence of the *Frzilator* feedback loop has not yet been shown experimentally, so in this chapter an alternative model of how the Frz pathway modulates reversals is proposed. The goal was to create an oscillatory mechanism that does not rely on a hypothetical feedback loop. Rather than focussing on the core Frz network, the *Motilator* considers downstream events involving the oscillation of motility-engine components.

Recent work by Mauriello et al. [99] has shown that FrzCD also appears to migrate around the cell and is not localised to the cell poles. By separating the *Motilator* part from the core Frz network, the system allows more complex models, that could also include FrzCD migration, to be created in the future. Furthermore FrzCD activation is not localised in the model, allowing for multi-site signalling should it later be shown that C-signalling can occur over the whole cell body.

The following physical interpretation of the *Motilator* is proposed. For simplicity cells have a region at each pole into which a finite amount of FrzS can migrate and accumulate with a migration rate mediated by MglA. MglA activity is in turn governed by FrzZ activity. To facilitate migration, there are two transport mechanisms operating inversely to each other. Each moves FrzS from one pole to the other. This does not specify exactly how the transport mechanism functions, and it almost certainly simplifies reality.

Kaiser [77] suggests MglA deactivates the motility engines and FrzE is used as the reversal signal. The *Motilator* uses MglA to regulate FrzS migration, which correlates to engine activity, and therefore indicates when a cell will reverse. If MglA deactivates an engine then it will trigger FrzS to migrate to the pole with the new engine. It is even possible that FrzS is used by cells to differentiate between old and new engine assemblies so that the correct one is deactivated. FrzE null mutants were found to reverse approximately every two hours [18]. It has not been established whether this behaviour is due directly to the Frz pathway or stochastic changes in protein levels. If the effect is deterministic, the *Motilator* can be tuned so that it reverses every two hours in the absence of FrzE. If the effect is stochastic, a secondary pathway could be added to the core Frz network to stimulate reversal in the absence of FrzE.

Many components of the *Motilator* can be reversibly modified. FrzCD is regulated by methylation (most likely by FrzF), while FrzE and MglA can be reversibly phosphorylated. FrzZ, FrzG, RomR and AglZ (in addition to FrzE) contain two-component system receiver domains and are therefore likely to have phosphatase activity and to be regulated by reversible phosphorylation. This raises the intriguing possibility that FrzS may act as a phosphatase to cause the transport system to alter its behaviour. x_1 and x_2 each represent a protein (or more likely group of proteins) which controls migration in one direction through the cell. If FrzS does act as a phosphatase, it could plausibly dephosphorylate MglA-P. The dephoshorylated form of MglA could then act as the trigger to cause FrzS to be migrated out of a compartment. x_1 would represent the amount of MglA-P present and x_2 either the dephosphorylated form of MglA, or another protein with which MglA and FrzS interact.

The *Motilator* has a relatively small parameter space yet its response can be tuned to match biological observation (Figure 1(b) in [110]). The *Motilator* system has the same key features: an initial ten minute reversal period, which can be a varied through C-signalling, and slow and rapid phases for FrzS migration. The schema is also consistent with data from Berlemann et al. [11, 12] which showed that *frzF* mutants do not ripple. If FrzF is suppressed to zero in the *Motilator*, oscillations stop.

The two phases of FrzS migration led to the postulate that a biological switch that is triggered by the level of FrzS at the poles might be controlling migration. Bi-stable switching is observed in many cell signalling mechanisms [6, 44]. A similar mechanism for governing motility could exist in *M. xanthus*. Markevich et al. [98] concluded from studies into bi-stability that a switch can be constructed from the mutual *inhibition* of a protein through its interconversion between two forms (e.g. phoshorylated and unphosphorylated).

A consequence of using a bi-stable switch is robustness to noisy input signalling and a built-in refractory period. When the FrzS has fully migrated to one end of a cell, the switch resets. Just after the switch reset, the system moves into the "slow" phase (see Figure 6.9) where it is most stable. A large burst of MglA* signalling during this phase (see Figure 6.7(b)) does not perturb the reversal cycle. As FrzS begins to migrate once again, it eventually moves the switch to the "fast" phase (see Figure 6.7(c)), where it is more unstable and sensitive to signalling and can be made to switch faster and increase the reversal frequency.

Stevens and Sogaard-Anderson [169] note that certain FrzCD mutants can signal constitutively [18, 24] leading to cell hyper-reversals. This behaviour is captured in the *Motilator*; constitutively high signalling of the FrzCD component of the *Motilator* will lead to high expression of MglA which cause rapid migration of FrzS between poles.

The *Motilator* model is able to reproduce the re-localisation patterns of FrzS during reversal in myxobacterial cells, and explains how the modulation via C-signalling is sufficient to cause cells to form ripples. While the focus was on myxobacterial motility, the *Motilator* is sufficiently generic to allow extrapolation to other gliding bacteria. Although the model presented here omits some biological details out of necessity, it

provides a theoretical framework for exploring the biochemical circuitry underlying cell reversal and spatial pattern formation.

Chapter 7

A Monte Carlo approach to modelling cell dynamics

Chapter 6 described the *Motilator* and explained in detail how the C-signalling mechanism might function and allow cells to ripple. The major focus of this chapter is building upon the *Motilator* and examining how the physical properties of the cell contribute to rippling. A secondary focus is the extension of the *Motilator* to examine the role nutrition plays in rippling and some of the issues with C-signal models that have hitherto gone unaddressed. The emphasis is on building a more realistic model of cell behaviour that can explain both rippling and streaming. Work from this chapter has been published as "Myxobacteria motility: a novel 3D model of rippling behaviour in *Myxococcus xanthus*" in *Communications of the Systematics and Informatics World Network* [60].

7.1 Introduction

Myxobacteria are fascinating members of the δ-proteobacteria, distinguished by a complex and social life-cycle involving multicellular development [35, 37]. In response to starvation, cells pass through several developmental stages over a 72 h period, culminating in the formation of *fruiting bodies* within which dormant cell-types called myxospores form. The first of these stages is the *Rippling* phase, occurring approximately 4 h into starvation. The population self-organises into mobile bands of cells, which reflect

off one another, giving the appearance of travelling waves. It is thought that rippling is an emergent property of an increased reversal frequency, brought about by the Frz cell-cell contact mediated signalling pathway. While much is known at the molecular level about components of the Frz pathway [205], the way in which it interacts with the machinery governing directional motion is poorly understood. Recent work has identified two polar proteins: FrzS and RomR, which dynamically relocate between poles during cell reversal [94, 110, 111]. Here, a model that incorporates physical cell dynamics and dynamic relocalisation of FrzS in response to Frz pathway activity is described. Surprisingly such a model describes many features of myxobacterial motility including oscillatory behaviour, fast and slow phases of protein relocalisation, a refractory period after reversal and robustness to variation in input signalling strength and duration.

7.2 Background

Myxobacteria are elongated, semi-flexible, rod-shaped cells that move in the direction of their long-axis, periodically reversing the direction of motion. The model myxobacterium (*M. xanthus*) is approximately 5 μm to 7 μm long and 0.5 μm in diameter. During vegetative growth cells move lengthwise reversing periodically, with a frequency of 0.104 reversals/min [71]. Under starvation conditions cells initiate a programme of multicellular development culminating in the formation of *fruiting bodies*, large aggregates of approximately 100,000 cells. Around 10 % of cells entering the fruit differentiate into myxospores (dormant cell-types), which rest quiescent until nutrients become available. The developmental process involves a series of macroscopic changes in colony morphology. A key regulator of development is C-signalling which occurs when C-signal, a cell surface-associated signal encoded by *csgA* is exchanged between cells in contact with one another. C-signal also stimulates the expression of *csgA* leading to positive feedback and a rise in C-signalling throughout development. Different colony morphologies are a consequence of different C-signalling levels [86]. C-signalling is thought to affect the reversal frequency of individual cells in a contact-dependent fashion [84, 86, 89].

In order to form ripples, C-signalling causes premature reversals allowing cells to synchronise their reversal cycle over time. Current understanding suggests that reversals are triggered by the Frz signal transduction pathway, which is homologous to the chemosensory (Che) system in *Escherichia coli* [39, 103]. While many of the proteins of the Frz pathway have been characterised, the molecular basis for Frz signalling is poorly understood. Recent evidence suggests one of the Frz proteins, FrzS is dynamically re-localised between cell poles. Thus the physical distribution of FrzS within a cell appears to be important in governing cell reversals [110].

Myxobacteria cells glide using the adventurous (A) motility system and the social (S) motility system [101]. S-motility is coordinated at the leading pole; cells extend type IV pili which can adhere to the surface of other bacteria or polysaccharides, and upon retraction the cell is pulled forward. A-motility is coordinated at the lagging pole. Cells extrude a slime which expands and generates a propulsive force to push cells forward [156, 196].

Myxobacteria cells are elongated and flexible [163] and this must, in part, account for their movement. A rigid geometry cell on a lattice is a very coarse approximation of this. Lattice models represent cells as a block of lattice nodes. Each cell has a rigid shape restricting its ability to move. On a LGCA lattice, each node has a finite number of channels connecting it to its neighbouring nodes. A cell must occupy one channel at a given node exclusively whilst it is on that node. Alber et al. use a hexagonal lattice with six channels (plus a rest channel) forcing cells to be orientated at a multiple of $pi/3$ radians at a given lattice point. If a cell changes direction it must do so by a minimum of $pi/3$ radians offering a very coarse and somewhat unrealistic approximation of cell movement. It was the the inherent limitations of such models that motivated the development an alternative more accurate model of rippling.

The *Motilator* explains how the Frz pathway could control an individual cell. In order to examine whether this is sufficient to get multicellular development, it is necessary to simulate a large population of cells in space as well as time and investigate how cells interact and how this affects the *Motilator*. The random nature of cell interactions means that C-signalling is both noisy and unpredictable. A cell may collide

with multiple cells at once leading to very high levels of signalling or it may glide in isolation and experience little if any external signalling. A model of the Frz pathway must be able to cope with all scenarios and still exhibit oscillatory behaviour.

As well as examining the biology of myxobacteria cells, it is also necessary to consider their physical characteristics since these play a role in controlling behaviour. Myxobacteria cells tend to move in straight lines during a reversal cycle. Recent work on modelling rippling has often approximated this behaviour by restricting cell movement, for example on a lattice [2, 20, 21] or simply direction based [5]. Chapter 6 used a cellular automata agent based model to demonstrate the effectiveness of the *Motilator* in a model where cells were restricted to moving in one of two directions as with previous studies [21]. This chapter investigates whether the *Motilator* correctly describes cell reversal behaviour sufficiently well, that in a model of freely moving cells which are unencumbered by artificial movement restrictions, rippling is still an emergent property of the cells. An extended version of the *Motilator* is incorporated into an agent based three-dimensional model to study rippling as the effect of cell biology and cell dynamics.

7.3 A revised model of the Frz transduction pathway

The Frz signal transduction pathway has been shown to play a key role in controlling cell motility [18, 204]. Current understanding of Frz pathway activity suggests that C-signal causes activation of FrzF [102, 157] which methylates FrzCD. Methylated FrzCD triggers autophosphorylation of FrzE which in turn phosphorylates FrzZ [205]. FrzZ-P then regulates MglA activity which is required for the appropriate localisation of FrzS and RomR.

Regulation of the Frz system is not fully understood; the nature of C-signal and its interaction on the Frz pathway is non-trivial. It has dual but conflicting roles which have yet to be reconciled. Once starvation induced behaviour is onset, C-signal amasses within a cell up until sporulation [86]. C-signal is necessary for rippling to commence; however, rippling also requires cells to increase their reversal frequency and amplification of C-signal causes cells to reverse less frequently [70]. Berleman et al. [11, 12]

provide evidence that rippling is a consequence of predation as well as C-signalling. FrzF is thought to be a homolog of CheR and FrzG a homolog of CheB [24] which methylate and demethylate FrzCD respectively. In the presence of nutrients FrzG is deactivated increasing the methylation of FrzCD leading to a higher reversal frequency. Shi et al. [148] also observed that very densely packed cells tend to reverse less frequently than natural wild type cells and it appears to be a result of affects on FrzCD. The nutrition model correlates with this behaviour this since a dense area of cells will consume nutrients faster forcing cells into the streaming and fruiting stages where they reverse more infrequently.

The circuit shown in Figure 7.1 is an extension of the *Motilator* (see Chapter 6) to incorporate the effects of FrzG. To summarise, the *Motilator* consists of two components: a representation of the Frz pathway coupled with a model of how Frz migrates between cell poles and is responsible for regulating cell reversal. Figure 7.1 illustrates how the components are connected.

The model represents how C-signalling controls reversals via MglA interacting with FrzS as follows:

$$\frac{d}{dt}f = \frac{(1-f)\cdot km_f}{K_f + (1-f)} - \frac{f \cdot kd_d}{Kd_f + f} \tag{7.1}$$

$$\frac{d}{dt}g = \frac{(1-g)\cdot km_g}{K_g + (1-g)} - \frac{g \cdot kd_g}{Kd_g + g} \tag{7.2}$$

$$\frac{d}{dt}c = \frac{(1-c)\cdot f \cdot km_c}{K_c + (1-c)} - \frac{c \cdot (1+g)\cdot kd_c}{Kd_c + c} \tag{7.3}$$

$$\frac{d}{dt}e = \frac{(1-e)\cdot c \cdot km_e}{K_e + (1-e)} - \frac{e \cdot kd_e}{Kd_e + e} \tag{7.4}$$

$$\frac{d}{dt}z = \frac{(1-z)\cdot e \cdot km_z}{K_z + (1-z)} - \frac{z \cdot kd_z}{Kd_z + z} \tag{7.5}$$

FrzCD (c) is regulated by FrzF (f) and FrzG (g) to be consistent with published data [11, 12, 169] showing that $\Delta FrzF$ mutants do not ripple and $\Delta FrzG$ mutants hyper-reverse. FrzG demethylates FrzCD and in its absence FrzCD output will be significantly higher translating into higher FrzZ and a higher reversal frequency.

The second part of the *Motilator* is left unchanged from my original study:

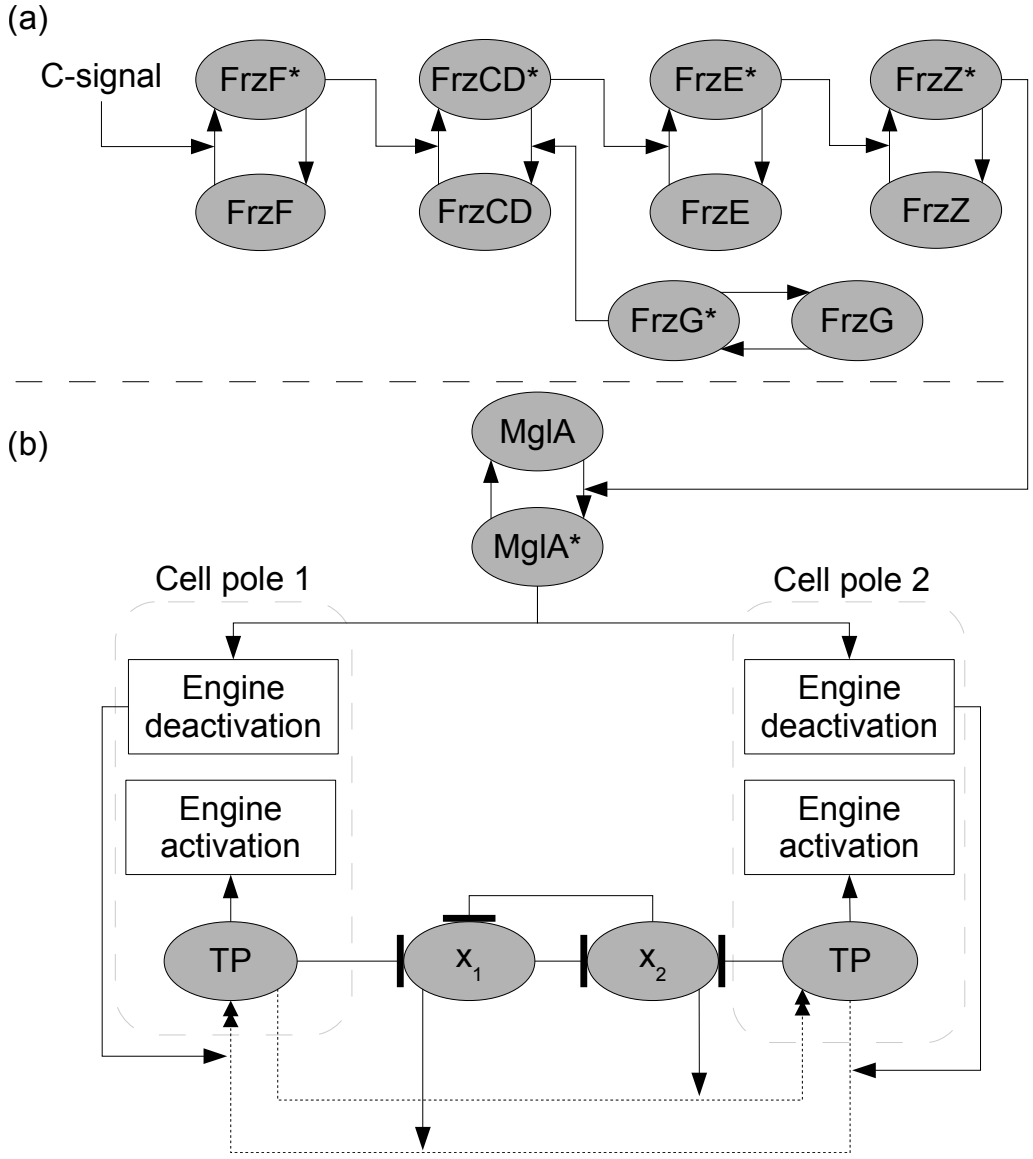


Figure 7.1: Schema of the *Phys-Motilator*. The effects of nutrition are considered via the effect of FrzG on FrzCD. Dashed lines with double arrowheads represent protein transport mechanisms and ovals represent proteins. Where proteins have an inactive and active form, an asterisk (*) denotes the activated form of the protein. (a) The core Frz network which responds to C-signalling via FrzF. (b) The Motilator, a potential mechanism to explain FrzS protein translocation. A biological switch regulates the flow of protein between pole one and pole two (which can be considered the head and tail of a cell). x_1 controls an active transport mechanism from pole two to pole one and x_2 controls an active transport mechanism from pole one to pole 2. Recruitment of FrzS at a pole is driven by MglA; however, the build up of FrzS eventually triggers the transport mechanisms to start moving FrzS to the other pole.

$$\frac{d}{dt}m = \frac{(1-m)\cdot z \cdot km_m}{K_m + (1-m)} - \frac{m \cdot kd_m}{Kd_m + m} \tag{7.6}$$

$$\frac{d}{dt}s_1 = \frac{x_1 \cdot m \cdot (1-s_1)}{K_{s1} + (1-s_1)} - \frac{x_2 \cdot m \cdot s_1}{Kd_{s1} + s_1} \tag{7.7}$$

$$\frac{d}{dt}s_2 = \frac{x_2 \cdot m \cdot (1-s_2)}{K_{s2} + (1-s_2)} - \frac{x_1 \cdot m \cdot s_2}{Kd_{s2} + s_2} \tag{7.8}$$

$$\frac{d}{dt}x_1 = \frac{\alpha_1}{\beta_1 + x_2^p} - x_1 \cdot (d_1 + s_1 \cdot k_1^{max}) \tag{7.9}$$

$$\frac{d}{dt}x_2 = \frac{\alpha_2}{\beta_2 + x_1^q} - x_2 \cdot (d_2 + s_2 \cdot k_2^{max}) \tag{7.10}$$

where m is the fraction of MglA*, s_1 is the level of FrzS at pole 1 and s_2 the level of FrsS at pole 2, x_1 controls the migration from pole 2 to pole 1 and x_2 controls the migration from pole 2 to pole 1. x_1 and x_2 have reciprocal feedback between each other forming a bistable switch. The parameters used to setup the *Phys-Motilator* are given in Table 7.1.

7.4 An off-lattice approach

One of the challenges in computational modelling is the trade off between realism and computability. Typically the more realistic a model is, the more computation is required and there is a corresponding reduction in what can be simulated.

Previous studies of rippling and C-signal dynamics have often used lattice based models [4, 167, 168]; however, there are a number of limitations to these modelling techniques which will be addressed here. Lattice models typically represent a cell using either a single node or a cluster of nodes. Myxobacteria cells are rod shaped and semi-flexible. Using the single node approach disregards cell shape but C-signalling and the motility machinery are localised at cell poles and a single node cell model cannot capture this very elegantly. Using the cluster of node approach can give cells a more realistic body shape, but it does not cope with cell flexibility very well. Cells can only move between other node locations and can only orient themselves with a finite number of degrees of freedom depending on the mesh type.

Table 7.1: Parameter values for the *Phys-Motilator* models. Where applicable, parameters were kept the same in the nutrient free and nutrient usage models.

Parameter	Value	Description
k_f, k_g, k_c, k_e, k_z	3.0 min^{-1}	
kd_f, kd_g, kd_c, kd_e, kd_z	1.0 min^{-1}	
K_f, K_g, K_c, K_e, K_z, Kd_f, Kd_g, Kd_c, Kd_e, Kd_z	1.0	Dimensionless Michaelis-Menten constant.
a_1	3.0	Governs the production of x_1. Dimensionless.
a_2	3.0	Governs the production of x_2. Dimensionless.
β_1	1.0	Dimensionless.
β_2	1.0	Dimensionless.
p	3.0	Co-operativity between x_1 and x_2.
q	3.0	Co-operativity between x_2 and x_1.
k_m	3.0 min^{-1}	
K_{d_m},K_m,K_m	1	Dimensionless Michaelis-Menten constant.
k_1,k_{d_1},k_2,k_{d_2}	1.0 min^{-1}	
K_1,K_{d_1},K_2,K_{d_2}	0.005	Dimensionless Michaelis-Menten constant.
d_1	1.0 min^{-1}	Minimum degradation constant for x_1.
d_2	1.0 min^{-1}	Minimum degradation constant for x_2.
k_1^{max}	0.8 min^{-1}	
k_2^{max}	0.8 min^{-1}	
n_d	0.000625	Dimensionless normalised rate of nutrient depletion. Only applicable to the nutrient usage model.

Myxobacteria cells tend to move along fairly straight line paths in the direction of their long axis. CA models often approximate this behaviour by restricting cells to moving in only one plane, effectively creating multiple independent streams running parallel to each other. Rippling patterns form using such models; however, an off-lattice approach allows cells to move without restriction, limited only by their physical characteristics rather than the artificial limitations of the lattice and can indicate whether observed patterns are due to anisotropic effects or a result of emergent behaviour.

Cellular Potts Models (CPM) [49] have been used as alternative way of modelling biological entities due to their ability to simulate physical bodies more realistically. By examining the Hamiltonian of a system, a measure of its energy, and using a Metropolis algorithm [27] to propose state changes, a system can be made to relax towards its equilibrium state which, if the Hamiltonian is specified correctly, should reflect the behaviour of a real system. Glazier et al. [49] and Starruß et al. [165] use CPMs on a regular lattice and use the Hamiltonian to control the motion, size and shape of each cell. Although this method allows very fine control over numerous physical properties of a cell, the disadvantage comes from the potentially large number of state changes that must be evaluated per iteration which scales poorly as the simulation size increases.

Principle physical characteristics of *M. xanthus*

- Rod shaped cells.
- Cells have a semi-flexible body.
- A-motility and S-motility machinery required for motility.
- Cells lay slime trails.
- Cells follow slime trails left by others.
- Cell functionality is localised, typically at the cell poles.

Figure 7.2: Principle physical characteristics of *M. xanthus* cells. The features outlined here are considered to play an important role in cell behaviour and should be described in a model of myxobacteria cells.

The literature on myxobacteria suggests there are a number of key physical properties that cells have which are important in rippling. Figure 7.2 summarises the important physical characteristics that a model of cell dynamics should consider to

accurately simulate cell movement. To overcome the geometric constraints a lattice based model places on movement, an off-lattice model coupled with the *Motilator* and a Monte Carlo algorithm [115] was adopted to study rippling since it allows cell properties, such as bending and stretching, to be modelled in a highly detailed way. This combined model will be referred to as the *Phys-Motilator* hereafter.

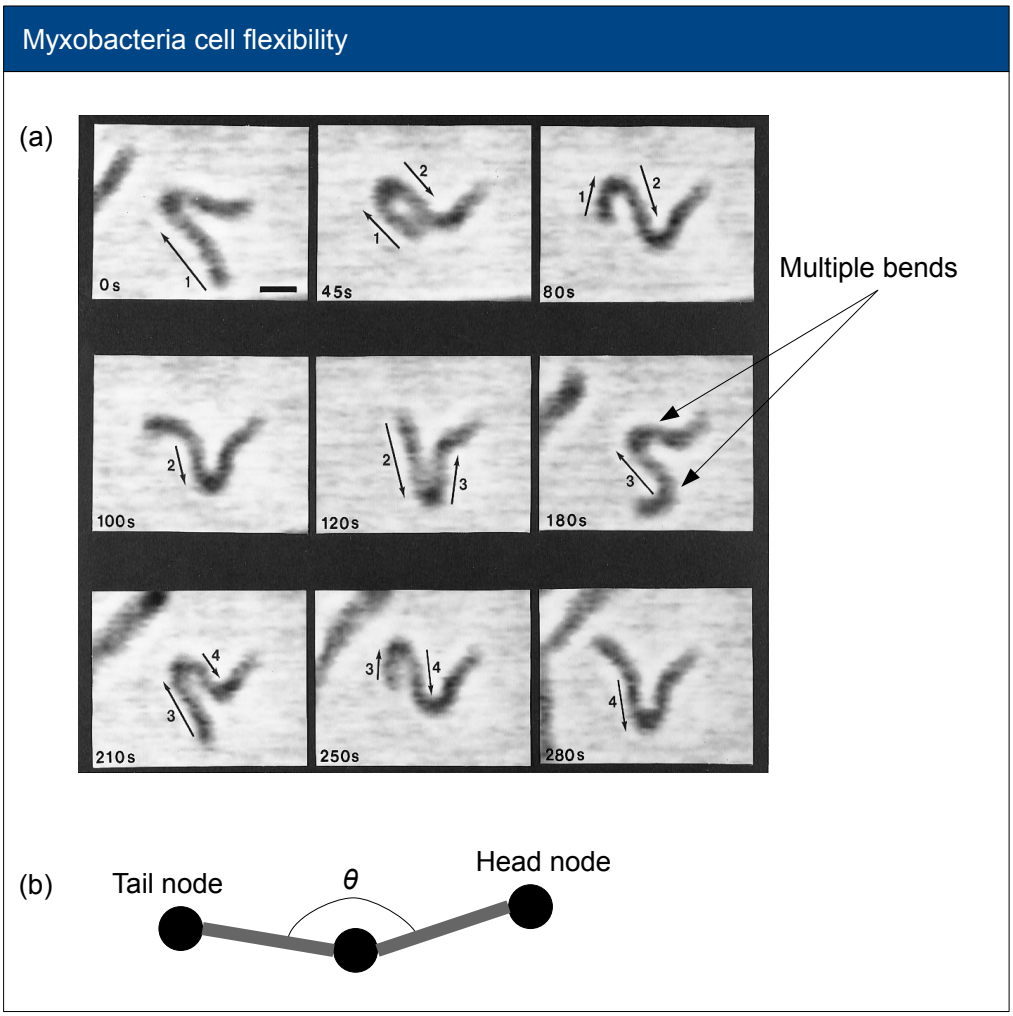


Figure 7.3: Myxobacteria cell flexibility. (a) Cells can flex and bend at multiple points along their cell length. Image adapted from (163). (b) The cell model proposed by Wu et al. (200) defines a cell as three connected nodes with a pivot point around the centre node.

Wu et al. [200, 201] use Monte Carlo techniques to investigate myxobacteria swarming behaviour. Their approach differs from Starruß et al. in that each cell segment is a fixed size and the medium cell is not considered, which greatly reduces the number of flips to compute at the expense of cell shape realism. An approach similar to this

was adopted in the *Phys-Motilator* since it allows a much greater number of cells to be simulated on a given workstation. Implementation details of the Wu et al. model raises a number of questions. Each cell consists of three linked nodes, effectively two rods that can pivot around a centre joint (see Figure 7.3(b)). A myxobacteria cell is capable of flexing and bending at multiple points along its length (see Figure 7.3(a)) but the three node model cannot capture this since cells can only bend at one point and doing so significantly alters the cell shape. There is no physical entity representing the cell volume except for the three nodes which are separated by a variable distance. In order to maintain a cell length to width ratio of 7:1, the gap between nodes appears to be ample for another cell to pass through. It is unclear from their work how cells passing through each other is dealt with.

The *Phys-Motilator* deals with the flexibility issue by constructing each cell from multiple segments so there are multiple pivot points. Each cell consists of N segments each with a finite, fixed volume to give the cell volume and shape (see Figure 7.4). Note that if the volume is equal to one, the model is analogous to the Wu et al. model [200]. A partial overlap is allowed between volumes to maintain a contiguous cell volume so that cells cannot pass through the space between adjacent segments. Cells may also partially overlap into other cell volumes although they may never occupy the same space as the segment centre of another cell. The two end segments, $n = 1$ and $n = N$ are nominally labelled as being either the head or the tail to give a cell an orientation. There are $N - 1$ connections between consecutive segments so they can be arranged to form the cell shape. The physical characteristics of a cell are determined by the physical properties of the connections.

Cells are positioned using a real coordinate system in a three-dimensional environment. A hybrid approach is adopted to represent cell orientation. Each segment is restricted to being in one of a finite number of orientations. This allows rotation angles to be pre-computed rather than during each iteration, saving a lot of time since the calculations are computationally costly. A simulation has a mapping between orientations and a vector field representing every possible direction a cell can move in from a given point. The number of possible orientations depends on the desired level of

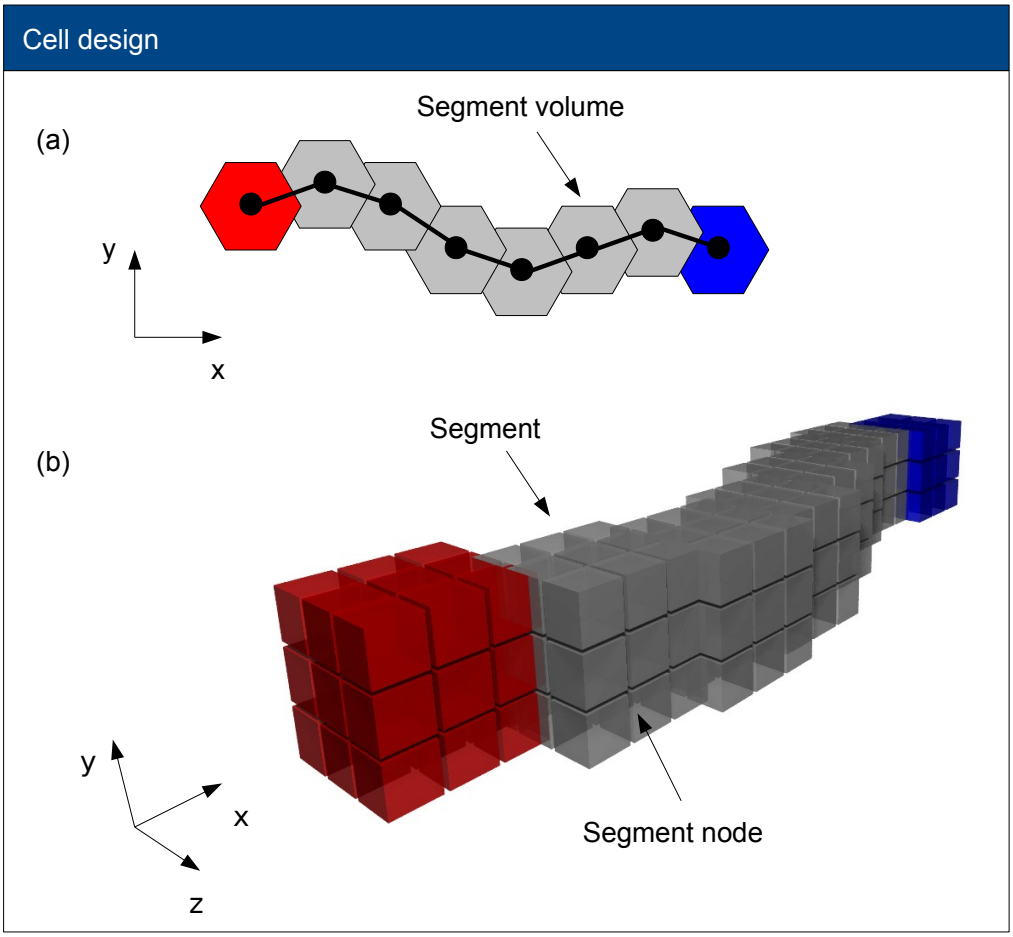


Figure 7.4: Schema of a segmented model cell. Each cell is composed of a number of connected segments, which can be of any size and shape. (a) two-dimensional view of cell. Each segment consists of a centre (black dot) and a neighbourhood of lattice nodes defining the segment volume (in this case hexagons). Neighbouring segment centres are a distance d from each other. (b) Three-dimensional view of a cell where each segment is a cuboid. Each cell has eight segments: a head (red), a tail (blue) and six body segments. Each segment comprises 27 segment nodes in a cube formation. Segments move independently allowing the cell body to be flexible. Overlap between segments allows the cell to maintain a continuous cell volume.

accuracy; if the number is very large, movement approximates a truly real coordinate system or, if the number is small, cells are restricted to moving as if they were on a lattice. If the orientations are restricted to $\{0, \pi/2, \pi, 3\pi/2\}$ for example, then cells move as if they were on a two-dimensional square lattice. This approach has the benefit of the increased freedom of motion of a real coordinate system but with a performance closer to a lattice based model.

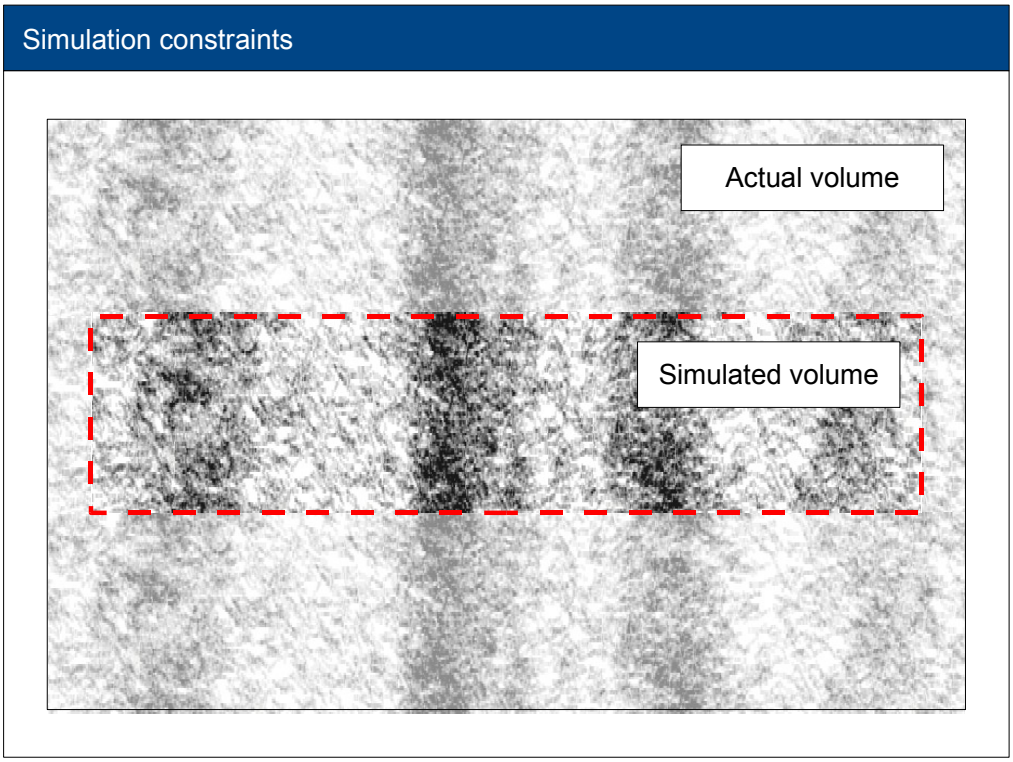


Figure 7.5: Simulation volume constraints. A rippling colony contains many thousands of cells. Simulations are only capable of modelling a portion of a colony; however, ripples are ordered so we can take a slice through the volume of a ripple region and simulate it without losing the spatial dynamics of the ripple.

It should be noted that there are computational resources are finite and it is not possible to model every cell in a large colony. A rippling population may contain hundreds of thousands of cells so there is a trade off between the complexity of the model and the number of cells that can be simulated. We can exploit the fact that ripples are fairly uniform in shape and cells maintain a spatial cohesion between themselves and adopt a strategy of simulating a volume within a colony (Figure 7.5). Periodic boundary conditions are used to simulate the effect of the volume being surrounded by

other cells forming part of a larger ripple. The model was implemented using FABCell as described in Section 6.5.

7.4.1 A multi-component model approach

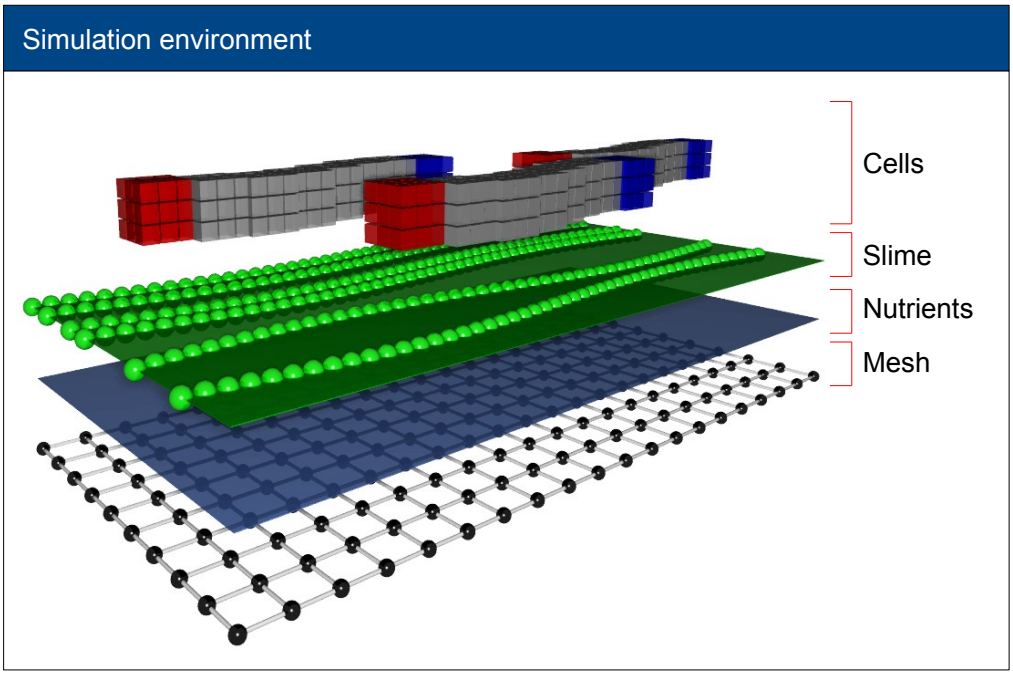


Figure 7.6: A simulation consists of an object stack representing each entity and feature present in the model. Every simulation uses a lattice to keep track of entities (even if objects do not explicitly use the mesh). Nutrient objects can be used to track any nutrients present. Slime objects keep track of slime trails deposited by cells. The cell object consists of cells in moving in a three-dimensional environment.

Figure 7.6 illustrates the components involved in a simulation. A typical simulation combines multiple objects organised into a hierarchy to represent all of the information about a system. This modular approach makes it easier to adapt models by adding new components which can all be accessed through a common set of interfaces. A more monolithic design would necessitate creating custom environments for each new model feature.

7.5 Simulating cell physics

The *Phys-Motilator* models how the Frz signalling pathway controls cell behaviour. This can be coupled to a model of *M. xanthus* cells to describe their physical charac-

teristics. Figure 7.7 shows the segmented cell model and the forces and properties that govern cell motion.

Table 7.2: The role of specific energy terms in the myxobacteria rippling Hamiltonian.

Term	Description
E_{align}	Myxobacteria cells align with other cells due to the effects of S-motility [7].
E_{bend}	Myxobacteria cells can bend and flex [163].
$E_{climbing}$	Myxobacteria cells can climb over each other when they are densely packed [31].
$E_{collision}$	Cells cannot occupy the space of other cells so this must be accounted for.
$E_{gravity}$	Cells are subject to the effects of gravity.
$E_{propulsion}$	Myxobacteria cells use the A-motility system to glide over a surface so require the ability to move independently of other cells [156].
E_{slime}	Myxobacteria cells can preferentially chose to follow [23].
$E_{stretch}$	Myxobacteria cells are not rigid and can therefore undergo small amounts of compression and expansion due to external forces.

In order to address some of the limitations of cellular automata models, namely the restricted movement and geometry of cells, an off lattice model coupled to a Monte Carlo algorithm was used to simulate cell colonies. The Monte Carlo algorithm evaluates the Hamiltonian of the system and uses stochastic sampling and rejection scheme to propose updates to the system state with energetically favourable changes being statistically more likely to be accepted. The models presented here are based upon and extend the work of Starruß et al. [165] and Wu et al. [200, 201]. The long term evolution of the system is therefore towards its lowest energy level reflecting the observed behaviour in a real world system. The following Hamiltonian function describes the energy of the system:

$$\begin{aligned}\mathcal{H} = {} & E_{align} + E_{bend} + E_{climbing} + E_{collision} \\ & + E_{gravity} + E_{propulsion} + E_{slime} + E_{stretch}\end{aligned} \tag{7.11}$$

Table 7.2 explains the role of each term and the functionality of each term is discussed below.

7.5.1 Alignment energy

In close proximity, cells tend to align with each other reflecting the effect of the S-motility engine. Cells extend Type IV pili from their leading pole which grab onto neighbouring cells. Upon retraction this pulls a cell closer to the neighbour it latched onto. The net effect of this is the alignment of cells.

$$E_{align}(a) = \alpha \cdot b_m \tag{7.12}$$

$$b_m = \begin{cases} \hat{\boldsymbol{c}} \cdot \hat{\boldsymbol{e}} & \text{if } (\hat{\boldsymbol{c}} \cdot \hat{\boldsymbol{e}}) \geq 0, \\ -(\hat{\boldsymbol{c}}_{\boldsymbol{m}} \cdot \hat{\boldsymbol{e}}) & \text{else.} \end{cases} \tag{7.13}$$

$$\hat{\boldsymbol{c}} = \frac{\boldsymbol{s}_{a,1} - \boldsymbol{s}_{a,N}}{\|\boldsymbol{s}_{a,1} - \boldsymbol{s}_{a,N}\|} \tag{7.14}$$

$$\hat{\boldsymbol{e}} = \frac{\boldsymbol{e}}{\|\boldsymbol{e}\|} \tag{7.15}$$

$$\boldsymbol{e} = \sum_{i \text{ neighbours}} \boldsymbol{e}_{i,1} - \boldsymbol{s}_{i,N} \tag{7.16}$$

where α is a dimensionless alignment coefficient, $\hat{\boldsymbol{c}}$ is the normalised average direction of the cell, $\hat{\boldsymbol{e}}$ is the average direction of all the cells in a local neighbourhood surrounding cell a. b_m reflects that cells tend to turn through the acute angle to align with other cells in either direction.

7.5.2 Bending energy

Each cell has a semi flexible body which must maintain a certain stiffness otherwise the cell folds up upon itself. Bending energy ensures that the radius of curvature of a cell does not exceed a threshold causing the cell to flail uncontrollably.

$$E_{bend}(a) = \sigma \sum_{i=1}^{N} b_{a,i}^2 \tag{7.17}$$

$$b_{m,n} = \begin{cases} b_{m,n+1} & \text{if } n = 0. \\ b_{m,n-1} & \text{if } n = N. \\ d_{m,n} & \text{else.} \end{cases} \tag{7.18}$$

$$d_{m,n} = cos^{-1}\left(\hat{\boldsymbol{e}}_{\boldsymbol{m,n}} \cdot \hat{\boldsymbol{f}}_{\boldsymbol{m,n}}\right) \tag{7.19}$$

$$\hat{\boldsymbol{e}}_{\boldsymbol{m,n}} = \frac{\boldsymbol{e_{m,n}}}{\|\boldsymbol{e_{m,n}}\|} \tag{7.20}$$

$$\boldsymbol{e_{m,n}} = \boldsymbol{s_{m,n+1}} - \boldsymbol{s_{m,n}} \tag{7.21}$$

$$\hat{\boldsymbol{f}}_{\boldsymbol{m,n}} = \frac{\boldsymbol{f_{m,n}}}{\|\boldsymbol{f_{m,n}}\|} \tag{7.22}$$

$$\boldsymbol{f_{m,n}} = \boldsymbol{s_{m,n}} - \boldsymbol{s_{m,n-1}} \tag{7.23}$$

where σ is a dimensionless bending coefficient, $d_{m,n}$ returns the angle between $\boldsymbol{e_{m,n}}$ and $\boldsymbol{f_{m,n}}$, $\boldsymbol{e_{m,n}}$ is the average direction of segment n of cell m and $\boldsymbol{f_{m,n}}$ is the vector between the segment ahead of n ($\boldsymbol{s_{m,n-1}}$) and the segment behind ($\boldsymbol{s_{m,n+1}}$).

7.5.3 Climbing energy

In areas of very high cell density, myxobacteria cells can climb over blocking cells. Each cell checks a local three-dimensional neighbourhood to see how many cells it is interacting with since the higher this number, the more likely it is the cell will stall due to an obstacle and need to climb. Climbing is dependent on space above the cell being available.

$$E_{climbing}(a) = -\eta\left(\text{cells}(\boldsymbol{b}, \boldsymbol{c}) \cdot \text{space}(\boldsymbol{d}, n)\right) \tag{7.24}$$

$$\boldsymbol{d} = \boldsymbol{b} + \boldsymbol{e} \tag{7.25}$$

$$\text{space}(\boldsymbol{f}, h) = \begin{cases} 1 & \text{if cells}(\boldsymbol{f}, h) = 0, \\ 0 & \text{else.} \end{cases} \tag{7.26}$$

$$\text{cells}(\boldsymbol{g}, h) = \sum_{\boldsymbol{i} \in h} \text{occupied}(\boldsymbol{i}) \tag{7.27}$$

$$\text{occupied}(\boldsymbol{j}) = \begin{cases} 1 & \text{if a cell occupies position } \boldsymbol{j}, \\ 0 & \text{else.} \end{cases} \tag{7.28}$$

where η is a dimensionless climbing coefficient, b is the current location of cell a, $\boldsymbol{c}$ is a neighbourhood, $\boldsymbol{d}$ a location a distance $\boldsymbol{e}$ above $\boldsymbol{b}$ to check for empty space, space($\boldsymbol{f}$, h) determines whether there is any space below the cell it can move into, cells($\boldsymbol{g}$, h) returns the number of cells occupying neighbourhood h centred around location $\boldsymbol{g}$ and occupied($\boldsymbol{j}$) returns whether a position $\boldsymbol{j}$ is occupied by a cell.

7.5.4 Collision energy

Each cell comprises a number of segments each with a finite volume. Segments exert a repulsive force between themselves to prevent cells colliding.

$$E_{collision}(a) = \tau \sum_{i=1}^{N} \sum_{(j,k)\ \text{neighbours}} \text{collision}(\boldsymbol{s}_{a,i}, \boldsymbol{s}_{j,k}) \tag{7.29}$$

$$\text{collision}(\boldsymbol{s}_{a,b}, \boldsymbol{s}_{c,e}) = \begin{cases} m & \text{if } \|\boldsymbol{s}_{a,b} - \boldsymbol{s}_{c,e}\| < d_{min}, \\ 0 & \text{else.} \end{cases} \tag{7.30}$$

where $\boldsymbol{s}_{a,b}$ is the position of segment b of cell a and d_{min} is the minimum distance allowed between segments of difference cells. The collision energy compares the distance between a segment and the neighbouring segments $\boldsymbol{s}_{j,k}$ around it and severely

penalises a cell for getting too close to another. Although the centres of segments cannot occupy the same space, a small overlap is allowed to model deformation effects of cells in close proximity. This is required because of the rigid segment shape which would otherwise not allow for this type of effect.

7.5.5 Gravitational energy

Cells are not allowed to climb indefinitely. They do so only if the local cell density is high enough. Once space becomes available cells will attempt to climb downwards back to the surface. Cells check whether there is space below them to move into and an energy penalty is applied if the cells attempts to move within its current plane.

$$E_{gravity}(a) = -\mu \left(\left[\hat{\boldsymbol{b}}_{\boldsymbol{a,1}} \cdot \hat{\boldsymbol{c}} \right] \cdot \text{space}(\boldsymbol{d}_{\boldsymbol{a,1}}, n) \right) \tag{7.31}$$

$$\hat{\boldsymbol{d}}_{\boldsymbol{m,n}} = \boldsymbol{s}_{\boldsymbol{m,n}} - \boldsymbol{e} \tag{7.32}$$

where μ is a sensitivity parameter, $\hat{\boldsymbol{b}}_{\boldsymbol{a,1}}$ is the normalised update direction of the head segment, $\hat{\boldsymbol{c}}$ a normalised direction vector pointing towards the ground, $\boldsymbol{d}_{\boldsymbol{m,n}}$ is a location below the centre of segment n of cell m and n is a local neighbourhood surrounding $d_{m,n}$.

7.5.6 Propulsion energy

The A-motility system provides myxobacteria cells with propulsion. Cells extrude a polysaccharide slime from nozzles at their lagging pole which expands when hydrolysed and pushes a cell forward. This effect is modelled using a propulsion term which causes cells to move preferentially in the average direction of the cell simulating the slime pushing a cell along.

$$E_{propulsion}(a) = -\epsilon \sum_{i=2}^{N} (\hat{\boldsymbol{u}}_i \cdot \hat{\boldsymbol{e}}) \tag{7.33}$$

$$\hat{\boldsymbol{e}} = \frac{s_{a,1} - s_{a,N}}{\|s_{a,1} - s_{a,N}\|} \tag{7.34}$$

$$\hat{\boldsymbol{u}}_n = \frac{s_{a,n-1} - s_{a,n}}{\|s_{a,n-1} - s_{a,n}\|} \tag{7.35}$$

where ϵ is a dimensionless propulsion coefficient, $\hat{\boldsymbol{e}}$ is the normalised average direction of the cell and $\hat{\boldsymbol{u}}_n$ is the update direction of segment n of cell a. Each segment moves towards where its head segment was previously except if this causes segments to become unaligned.

7.5.7 Slime trail following energy

As well as extruding slime to move, cells can also detect slime trails left by other cells and preferentially follow them [23]. This allows cells to follow other adventurous cells and leads to the formation of streams that can break away from the main colony. Slime following is complementary to A-motility. As each cell moves, it deposits a slime trail. This is a set of normalised vectors representing the average direction of a cell. The slime ages over time and is eventually removed. Cells can sense slime trails within a limited neighbourhood around them. Using a weighted sum of the all slime trail directions based upon their age, the average slime direction is calculated and cells will preferentially follow that. A weighted sum was used to account for that fact that a cell is more likely to follow a large slime trail than a small one. If a cell was to pick a random slime trail and follow it, it might end up moving perpendicular to the major slime trail which is unrealistic.

$$E_{slime}(a) = \phi\left(\hat{\boldsymbol{b}}_{\boldsymbol{a}} \cdot \hat{\boldsymbol{c}}\right) \tag{7.36}$$

$$\hat{\boldsymbol{b}}_{\boldsymbol{m}} = \frac{\boldsymbol{s_{m,1}} - \boldsymbol{s_{m,N}}}{\|\boldsymbol{s_{m,1}} - \boldsymbol{s_{m,N}}\|} \tag{7.37}$$

$$\hat{\boldsymbol{c}} = \frac{\boldsymbol{c}}{\|\boldsymbol{c}\|} \tag{7.38}$$

$$\boldsymbol{c} = \sum_{i \text{ neighbours}} \text{slime}(\boldsymbol{i}) \tag{7.39}$$

$$\text{slime}(\boldsymbol{m}) = \frac{\boldsymbol{t_m}}{\|\boldsymbol{t_m}\|} \tag{7.40}$$

where ϕ is a dimensionless slime following coefficient, $\hat{\boldsymbol{c}}$ is average direction of the slime trails in a neighbourhood and slime($\boldsymbol{m}$) is the normalised direction vector of the slime trail at location $\boldsymbol{m}$.

7.5.8 Stretching energy

Myxobacteria cells have a finite volume and fixed cell length (although a small amount of variation is allowed to account for external forces acting on a cell). These constraints are enforced with a term to measure a cell's stretching energy. This governs a cell's length and is analogous to the spring constant in Hooke's Law.

$$E_{stretch}(a) = \lambda \sum_{i=0}^{N-2} \left(\|\boldsymbol{s_{a,i+1}} - \boldsymbol{s_{m,i}}\| - d_0\right)^2 \tag{7.41}$$

where λ is a dimensionless stretching coefficient, N is the number of segments in cell a, d_0 is the optimal distance between segments, $\boldsymbol{s_{k,l}}$ is the vector position of segment l in cell k and λ a dimensionless stretching coefficient. Stretching energy is defined as a squared sum which compares the distance between the centres of neighbouring segments $\boldsymbol{s_{m,i}}$ and $\boldsymbol{s_{m,i+1}}$ to d_0 and penalises a cell for allowing segments to get either too close or too far apart.

7.5.9 C-signalling

Two cells are considered to be C-signalling between each other under the following circumstances: the cell poles are in close proximity (less than one cell width apart), cells are closely aligned but oppositely oriented and the distance between the heads of each cell or the head of one cell and the tail of the other is less than one cell width. Oppositely oriented is defined to be when the acute angle between the normalised average direction vectors of both cells is greater than $\pi/2$ radians.

$$s(x) = \zeta \sum_{i \text{ neighbours}} c_{x,i} \tag{7.42}$$

$$c_{x,y} = \begin{cases} 1 & \text{if } \boldsymbol{a_x} \cdot \boldsymbol{a_y} < 0 \\ 0 & \text{else.} \end{cases} \tag{7.43}$$

$$\boldsymbol{a_m} = \frac{\boldsymbol{s_{a,1}} - \boldsymbol{s_{a,N}}}{\|\boldsymbol{s_{a,1}} - \boldsymbol{s_{a,N}}\|} \tag{7.44}$$

where $s(x)$ returns the amount of signalling between a cell x and the cells in a local neighbourhood, ζ is a parameter controlling the sensitivity of signalling, $c_{x,y}$ takes the dot product between the average direction of cells x and y and $\boldsymbol{a_x}$ returns the normalised average direction vector of cell x.

7.5.10 Simulation parameters

The model was setup using parameters given in Table 7.3.

7.6 Results

The *Phys-Motilator* was used to create a number of simulations of cell behaviour. All models used the parameter set given in Table 7.3 except where the model was specifically altered to investigate a particular property. Each simulation consisted of 5500 cells in a three-dimensional simulation volume with periodic boundary conditions in the xz-planeand yz-plane. Each cell used the *Phys-Motilator* to dictate its behaviour.

Table 7.3: Parameters for the off-lattice simulation of *M. xanthus*.

Name	Value	Description
λ	3.0	Dimensionless stretching energy parameter.
α	2.0	Dimensionless volume energy parameter.
σ	12.0	Dimensionless bending energy parameter.
ϵ	2.0	Dimensionless propulsion energy parameter.
ν	1.0	Dimensionless climbing energy parameter.
μ	3.0	Dimensionless gravity energy parameter.
τ	100.0	Dimensionless collision energy parameter.
kT	0.3	Boltzmann constant $\times$ temperature.
c	5500	Number of cells.
V_s	0.422 μm^3	Segment volume.
d_0	0.75 μm	Target distance between adjacent segments ($\sqrt{V_s}$).
V	$250 \times 30 \times 30$ μm^3	Simulation volume.

Cells were initially randomly distributed and the system was allowed to evolve. There was no birth or death so the number of cells remained constant reflecting that during starvation cells do not divide so a rippling population will contain an approximately static cell population.

7.6.1 Motility

Cells exhibit both A-motility and S-motility characteristics depending on cell density (see Figure 7.7). Cells in close proximity will tend to align with each other forming local clusters with expand to form streams. Isolated cells will deposit slime trails whilst tending to follow slime trails left by other cells thereby allowing them to move out from a colony edge. It should be noted that density plots displayed here typically do not show slime trails as they tend to obscure spatial patterns formed by the cells.

7.6.2 Slime trails

Cells were biased towards following the newest and thickest slime trails as they are more likely to do so in reality. The alternative scenario would be for cells to scan the neighbourhood and select a time trail and follow it. This was felt to be unrealistic as it treats all slime trails as equal and would allow a cell to follow a very thin slime trail over a larger trail even though this is unfavourable and is unlikely to happen in a

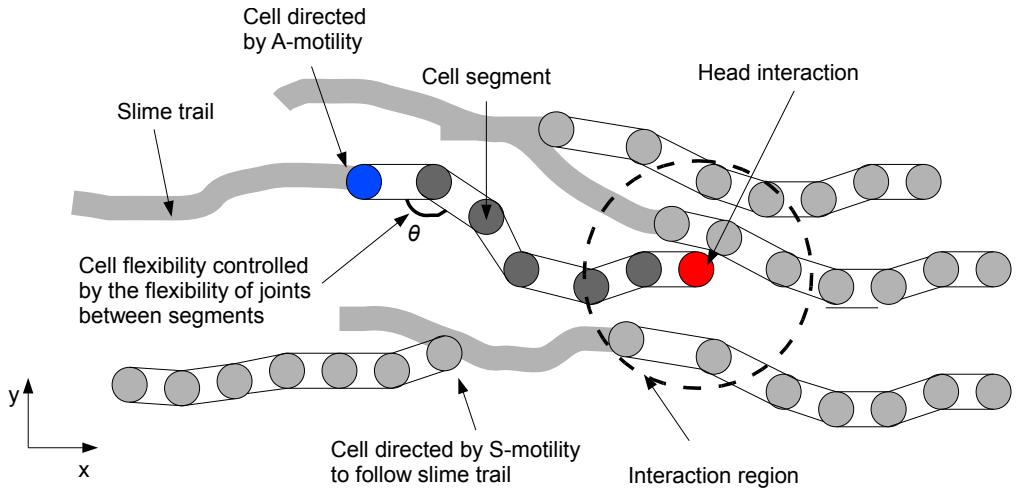


Figure 7.7: Physical characteristics of *Phys-Motilator* cells. Each cell consists of a head (red node), body (grey nodes) and tail (blue node). Cells move in the direction of the head. Cells are semi-flexible and can exhibit both social (S) and adventurous (A) motility. Cells deposit slime trails which other cells can preferentially follow.

colony.

The slime trail machinery was found to be important in maintaining spatial cohesion between cells. Figure 7.8 shows a comparison of two simulations: one where cells lay and follow slime trails and the other where the machinery is switched off. Ripples do not form when the slime machinery is switched off. Ripple formation appears to be strongly dependent on cell alignment, both within ripples and in troughs when cells are acting under A-motility. The slime serves to direct cells so in regions of low cell density they follow straight paths between ripple fronts. Without slime, cells move more randomly unless they are close enough to other cells for S-motility to cause alignment. Leading cells within a ripple front or densely populated area break away and expand from the pack in "Frizzy" like patterns as cell orientations become more diverse. Ripples break down as other cells follow the leading cells with S-motility. A-motility and S-motility are not sufficient by themselves to maintain cell order over long distances which seems to be a requirement of rippling.

7.6.3 The effects of slime trail following

Figure 7.9 is a close up view of a ripple front showing the effects of slime trail following. A and S motility are sufficient to allow short range spatial cohesion but not to allow

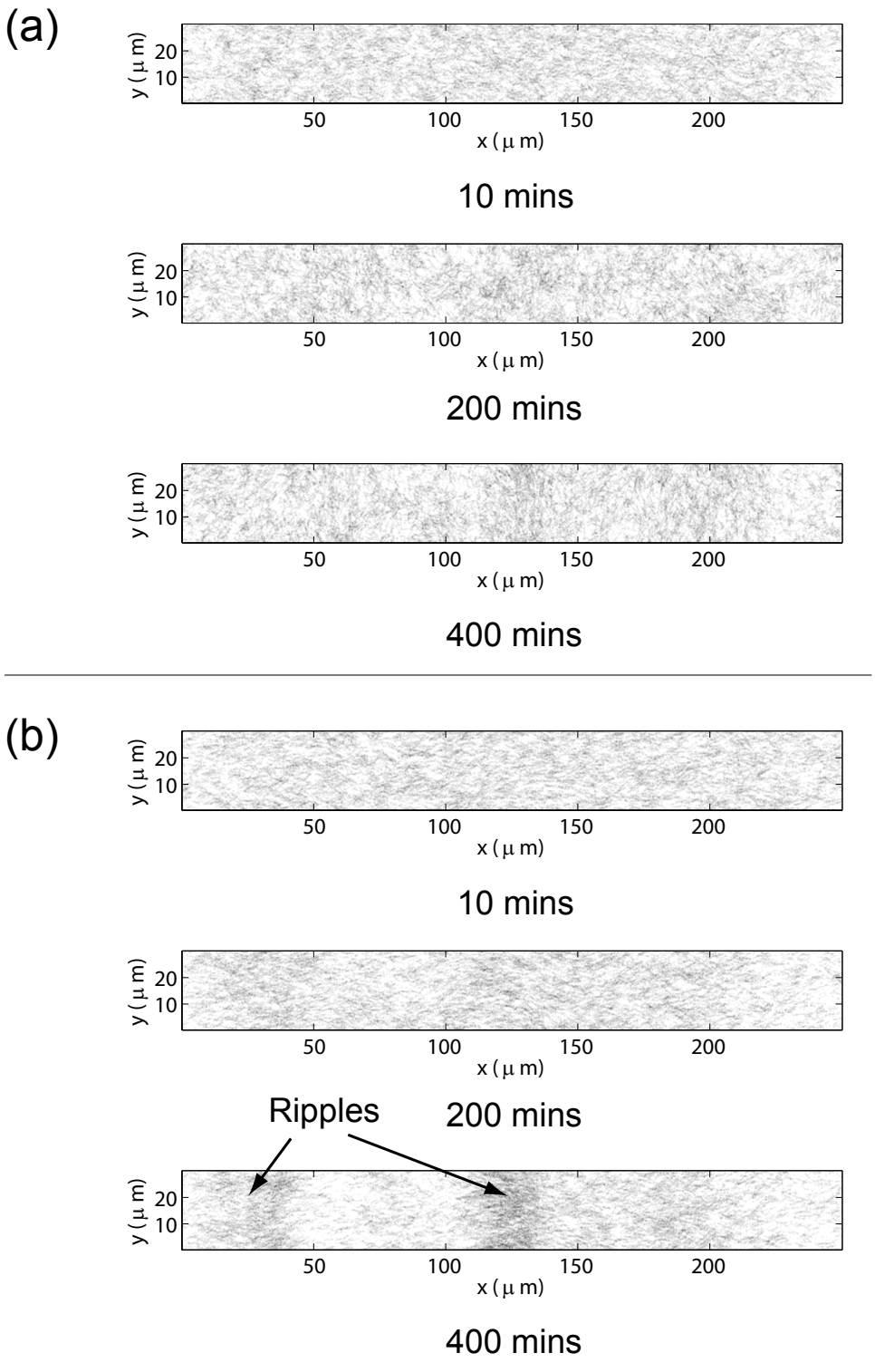


Figure 7.8: Comparison of the ability of the *Phys-Motilator* to form ripples when slime trail following is disabled and enabled. (a) Slime trail following disabled. Without the slime engine, cells do not maintain spatial cohesion sufficiently for ripples to form. (b) Slime trail following enabled. By 400 min, cells have already started to form ripples.

cells to remain aligned in the ripple. Lead cells can break away from the pack, directing their local neighbours to follow them, forming small clusters of aligned cells. Globally the ripple breaks down as cells migrate away and "Frizzy" like patterns start to form as cells spread out. Slime trail following is required so that cells remain aligned in much bigger groups allowing the cell body to remain aligned for much longer and the ripple to persist.

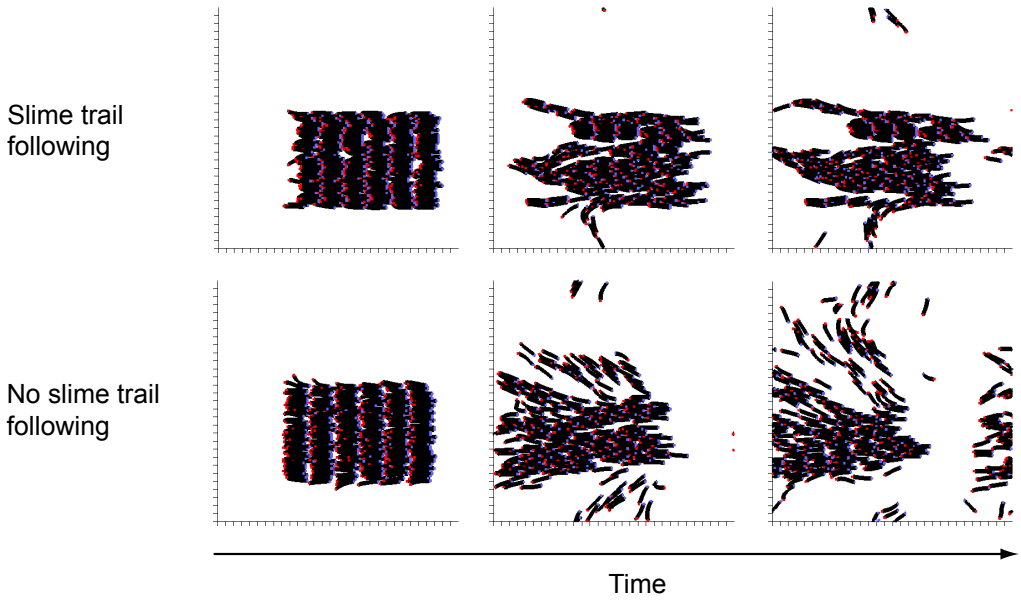


Figure 7.9: The effects of slime trail following on local cell alignment in a ripple. Identical simulations of a cell population were performed where slime trail following was switched on or off. Results for the two experiments are shown at the same time points. following A and S motility are sufficient for short range spatial cohesion but not long range; cells align into smaller clusters but globally the ripple breaks down. Slime trail following helps cells maintain long range cell alignment allowing ripples to persist (see Figure 7.8).

7.6.4 The effects of climbing

The models start initially with a monolayer of cells however it was found that if cells are tightly packed together, ripples do not form if the strict monolayer of cells is maintained. Cells have little room to manoeuvre and are frequently obstructed by other cells leading to stalling. Cells towards the periphery of a colony are able to move around obstacles but this is dependent on enough space being available to allow a change of course. The majority of cells are very close to each other and cannot avoid collisions

without some kind of severe and unrealistic body deformation. Models where cells are restricted to moving in fixed planes or with fixed geometries [21, 154] do not address the issue of how large numbers of elongated, semi-flexible rod shaped cells interact.

Ripples are not a consequence purely of cells colliding, blocking each other and then reversing as tracking of individual cells showed they typically stalled for approximately one minute [154]. During later stages of development there is a propensity for aggregation centres to form when cells are tightly packed; however, they are not a common artefact during rippling so there must be other factors maintaining rippling. Slime trail following coupled with S-motility and A-motility is sufficient to maintain ordered cells within a ripple but it does not prevent collisions and stalling.

Cells move around in a slime matrix and certainly during fruiting body development, they have the ability to climb over each other. The effects of allowing cells to climb within a rippling model was studied. Figure 7.10 shows a comparison of the effects of having climbing enabled and disabled in the *Phys-Motilator*. It was immediately apparent that climbing plays an important role in ripple formation since without it, cells are completely unable to form ripples or other spatial patterns.

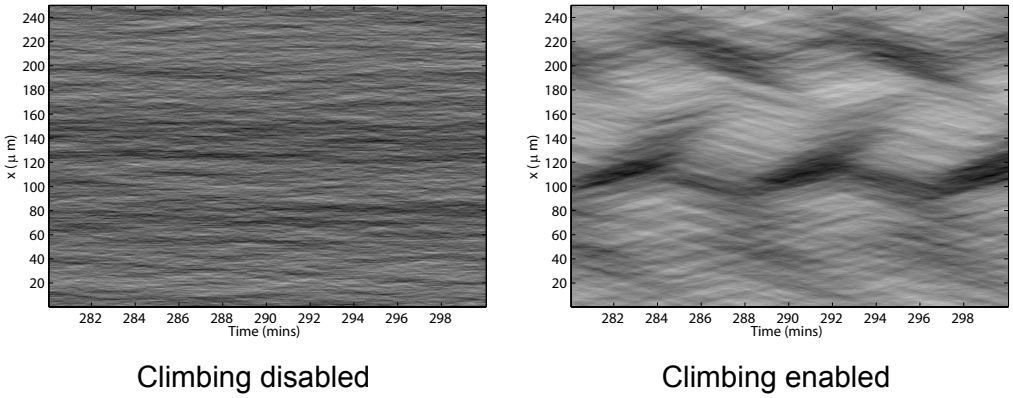


Figure 7.10: Space time plot showing the effects of cell climbing on ripple formation after 280 min. When cells can climb, characteristic ripple bands emerge; however, there is no spatial cohesion or pattern formation when cells cannot climb.

Figure 7.11 shows a comparison of how proximity (see Equation 6.21) varies over 300 min during a simulation run. When cells are allowed to climb, proximity increases indicating cells are getting closer together. Proximity is consistently lower in a cell population that cannot climb indicating cells are not on average getting closer and

there is no coordinated behaviour.

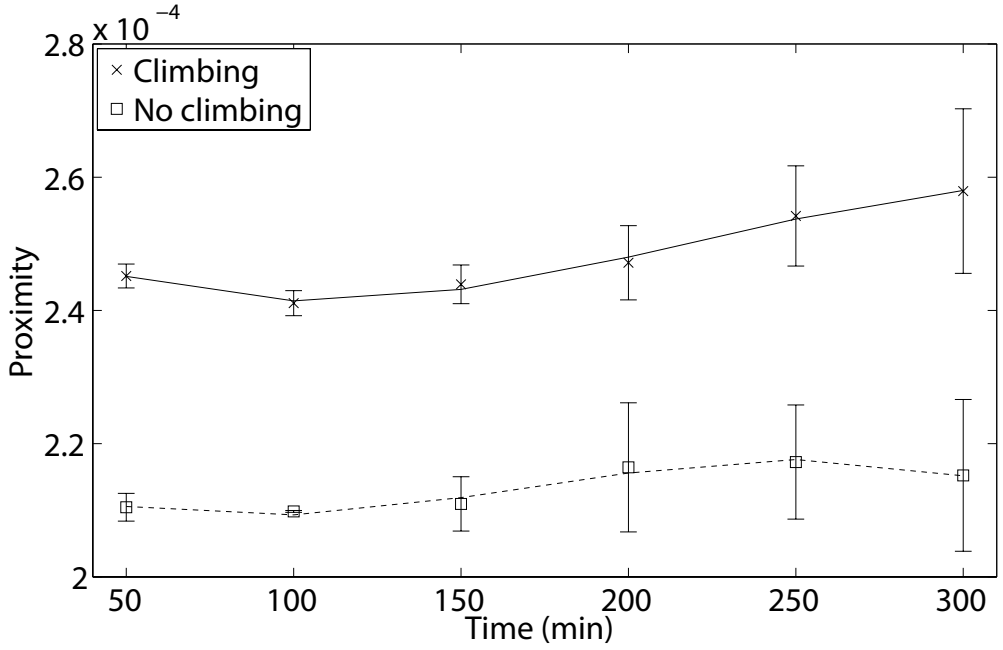


Figure 7.11: The effects of cell climbing on cell proximity during rippling over a 300 min interval. Proximity is measured from 50 min to allow cells to get into a relaxed state. Proximity is consistently higher when cells are allowed to climb and rises indicating cells are getting closer together and forming ripples. Proximity was calculated using the proximity function in Equation 6.21.

7.6.5 Collision avoidance

In a dense population of cells, cell motion can very easily become restricted leading to cells stalling and aggregation centres forming prematurely as cells get blocked. Clearly this does not happen in a rippling population as ripples persists so there must be other factors determining how cells move. S-motility and slime trails allow cells to remain aligned with each other and some flexibility within the cell allows them to bend to avoid obstacles; however, in a tightly packed region, there is not enough space for a cell to bend around on itself to move away from the obstacle. The cell could simply stall and wait for its motility machinery to reverse but Sliusarenko et al. [154] suggest that cells do not stall for long periods when rippling. The solution appears to be to allow cells a limited ability to climb over each other. This is dependent on the local cell density around a cell; an obstacle of one cell in front is not sufficient to trigger a cell to

climb, but the probability rapidly increases with cell density. A form of pseudo gravity is also present to stop cells climbing unnecessarily as it penalises a cell for trying to climb so it can only do so if it energetically favourable; other options such as bending require less energy so will be chosen preferentially.

7.6.6 Signalling and reversing

In the segmented cell model, only the head and tail region are sensitive to signalling as C-signalling is thought to occur only at the poles [72] and only when cells are colliding. This means the majority of the cell body is insensitive to signalling. Consequently the amount of signalling a cell receives diminishes. In the CA models such as Börner et al. this problem does not arise as the cells are either heads or tails and so are sensitive to signalling at all times. These models also simulate cell motion on a minute scale. The segmented model simulates cells on a second scale. It became apparent that the duration of a signalling pulse is important. Even in a dense population, a cell will not actually receive that many signalling events since only head-head interactions trigger C-signal exchange. Therefore cells must be very sensitive to signalling if only a few events are required to trigger reversal and the effect of signalling must last several seconds if it is to sufficiently perturb the reversal cycle and cause a premature reversal.

7.6.7 Ripple formation

The *Motilator* is sufficient to reproduce rippling in a CA model of cell formation (see Section 6.7.3) To determine the efficacy of the *Phys-Motilator* in organising semi-flexible, free moving cells, it was implemented in the off-lattice model using all of the properties specified by the Hamiltonian (see Section 7.5). To negate the effects of nutrition and assess rippling as a consequence only of C-signalling and body dynamics, a constant nutrient source was introduced so that the effects of FrzG were not considered.

Figure 7.12 shows the output of a rippling simulation. From an initial random distribution cells organise themselves into ripples after 400 min. Three main ripple bands form, each of which is sufficiently far from the others that the ripples do not actually collide with each other. It is apparent from Figure 7.13 that cells form into three

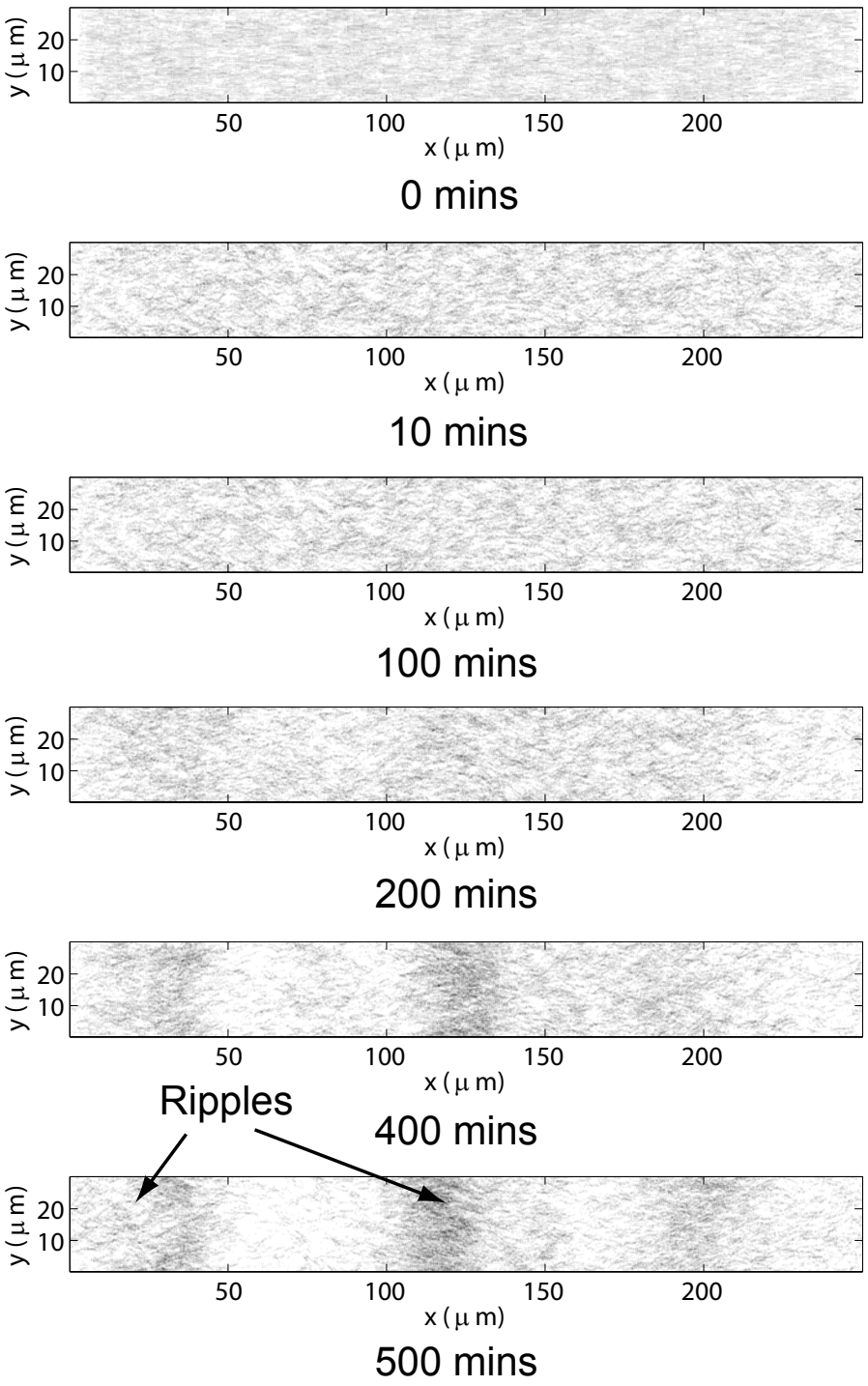


Figure 7.12: Ripple formation using an off lattice simulation of 5500 cells with an unlimited nutrient source. Time points correspond to those in Figure 7.13. Cells start from an initial random distribution. After 400 min, three distinct, stable ripple bands have formed.

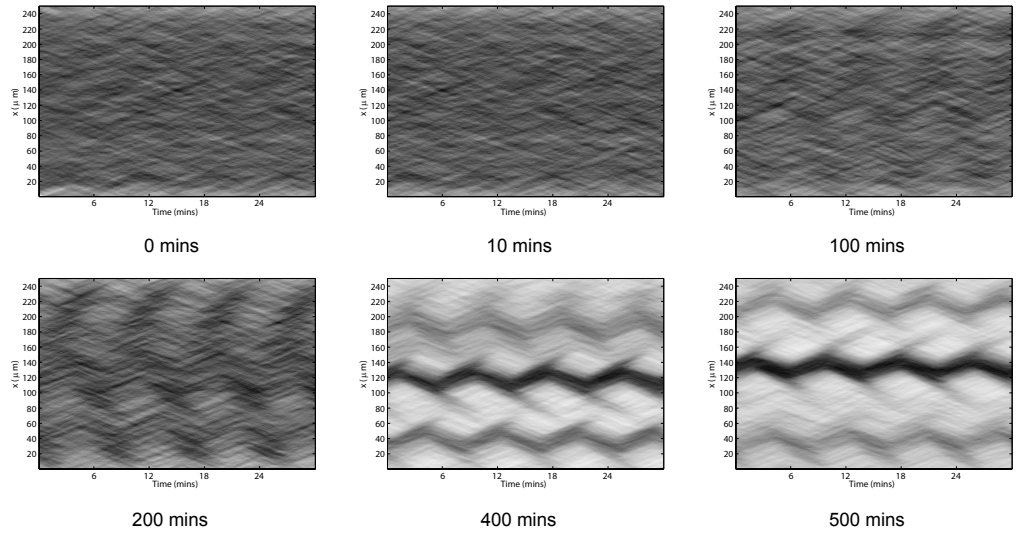


Figure 7.13: Space time plots of ripple formation using the *Phys-Motilator* with an unlimited nutrient source. Time points correspond to those in Figure 7.12. Cells start from an initial random distribution. After 400 min, three distinct ripple bands have formed, which persist indefinitely.

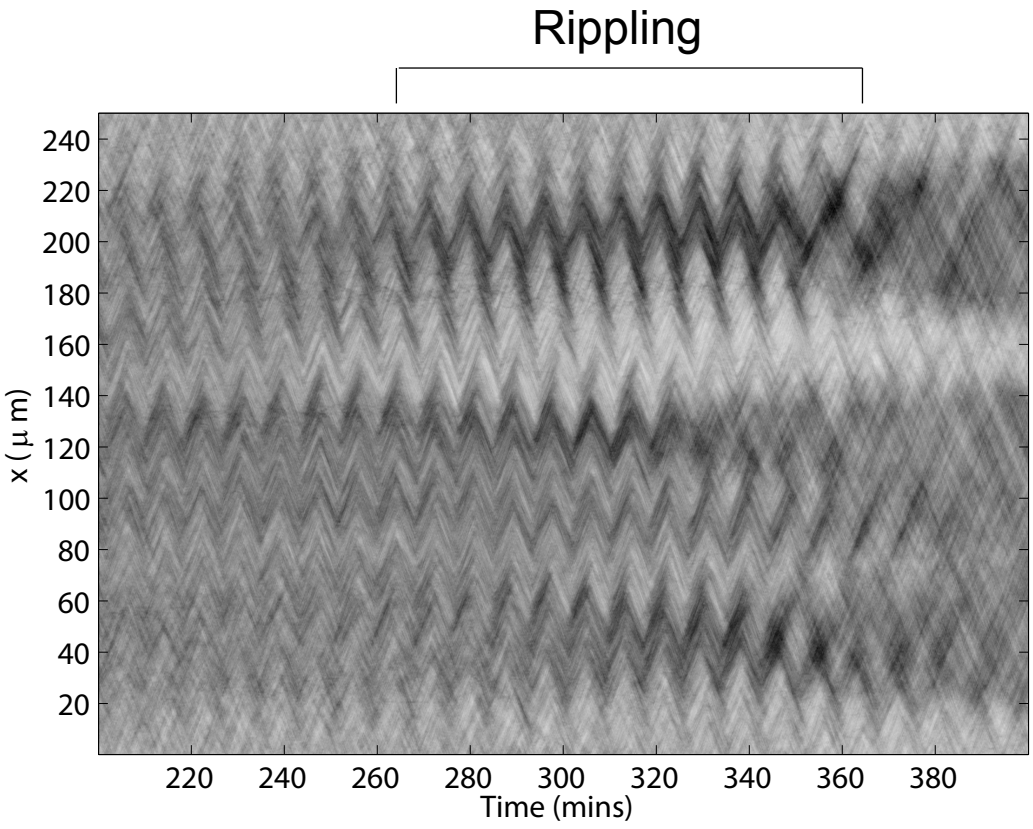


Figure 7.14: Space time plots showing the transition from pre-rippling to rippling. Ripple simulation of 5500 cells starting from an initial random distribution. Dark regions indicate dense areas of cells, light regions indicate areas of low cell density. Distinct ripples begin to form after 250 min and persist for 110 min before beginning to disband into streams.

non-overlapping bands of cells. The well defined oscillations indicate that once cells have joined a ripple they do not leave and the ripples are very stable and persistent if unperturbed. This simulation is available as Movie `mov_phys_mot` on the `CD-ROM` accompanying the thesis (see Appendix A). Figure 7.14 shows the transition of cells from a pre-rippling to a rippling state. Ripples evolve over time rather than through spontaneous formation.

In a cell dense environment, it might intuitively be expected that C-signalling would be high but two factors limit this: polar sensitivity and traffic jams. C-signalling is thought to only occur at the cell poles [72] meaning the majority of the cell body does not take part in signalling so that whilst cells may be in close proximity, there are actually only a few pole-pole interactions taking place. Traffic jams arise in a closely packed environment since there is limited space for cells to move and it is not always possible for cells to climb over one another. Once a jam forms, cells might get stuck temporarily until they either naturally reverse or the jam disperses. During this waiting time, there is again limited opportunity for signalling because the position of cell poles changes very little. These two factors ensure that at a given time, only a small percentage of cells will actually be close enough for their heads to interact. In CA models, which simplify cell physics so a cell is a single node entity on a lattice [21], this phenomenon is less apparent since each cell is either a head or a tail; with no insensitive region there is much higher signalling and a higher reversal rate so pattern formation is faster.

Figure 7.15 shows how the reversal times of cells change over a 500 min period. In the pre-rippling phase (0 min to 100 min), the majority of the cells are reversing after 2 min to 5 min. As ripples begin to form, the spread of reversal periods narrows and the average shifts towards 4.59 min. As cells synchronise their reversal cycle, C-signalling will decrease between patches of cells travelling in the same direction. There are fewer head-head interactions and so reversal frequencies decrease. At the same time, a large number of cells are not synchronised within a ripple and continue to C-signal, maintaining the average reversal frequency over the duration of the simulation; cells may be climbing over each other and freedom of movement allows cells to interact at

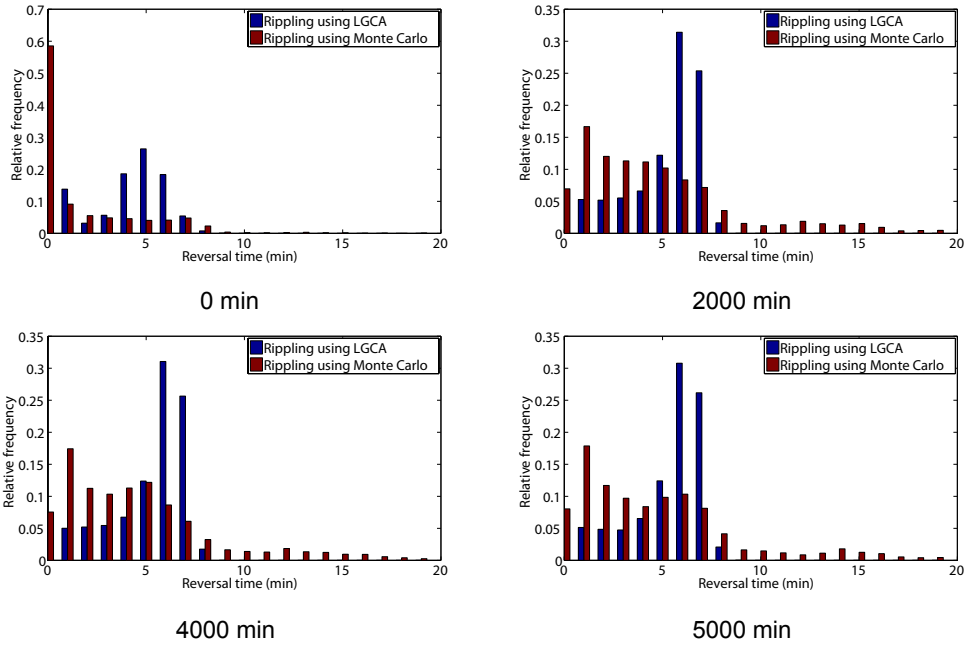


Figure 7.15: Histograms of cell reversal times during rippling. Profiles are shown for LGCA model described in Section 6.7.3 (blue bars) and the Monte Carlo method (red bars).

a range of orientations. Both the LGCA and Monte Carlo models have similar reversal distributions indicating the *Motilator* works consistently in the different models. The distribution of reversals is wider in the Monte Carlo model reflecting that the elongated cell shape allows for a greater variance in the number of collisions a cell experiences; the LGCA model restricts the number of cells a cell can collide with.

7.6.8 From rippling to streaming

To study the effect of the loss of nutrients upon the Frz pathway in the *Phys-Motilator*, a finite nutrient layer was added to the model (see Figure 7.6). The nutrient source was normalised to one at the start of the simulation and the system was tuned so it was depleted after approximately five hours to allow sufficient time for ripples to form. Where applicable, parameters were kept the same as in the constant nutrient model (see Table 7.1). Once the nutrients are exhausted, FrzG becomes up-regulated and demethylates FrzCD reducing the switching of FrzS between cell poles causing cells to reverse less frequently and migrate in one direction for extended periods (see Figure 7.1). Figure 7.16 illustrates the effect of the up regulation of FrzG on the *Phys-Motilator*. As

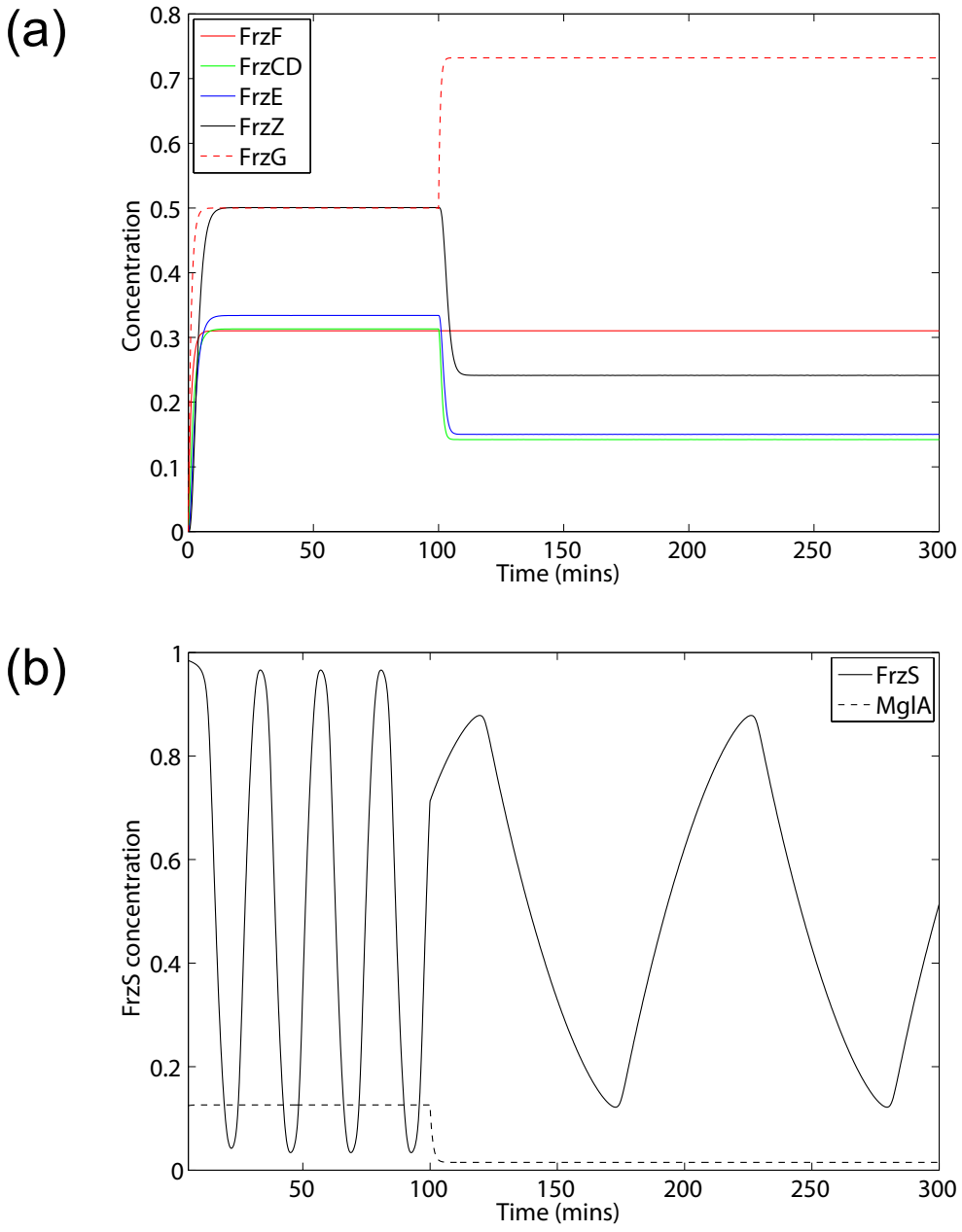


Figure 7.16: The effect of FrzG on the *Phys-Motilator*. (a) Response of the core Frz network. (b) Response of FrzS (coupled to the core Frz network via FrzZ and MglA) during the same time period.

FrzG increases it down regulates FrzCD, which in turn down regulates FrzE and FrzZ (see Figure 7.16(a)). The reduction in FrzZ causes a reduction in MglA production triggering a much slower translocation of FrzS leading to a decreased reversal frequency (see Figure 7.16(b)).

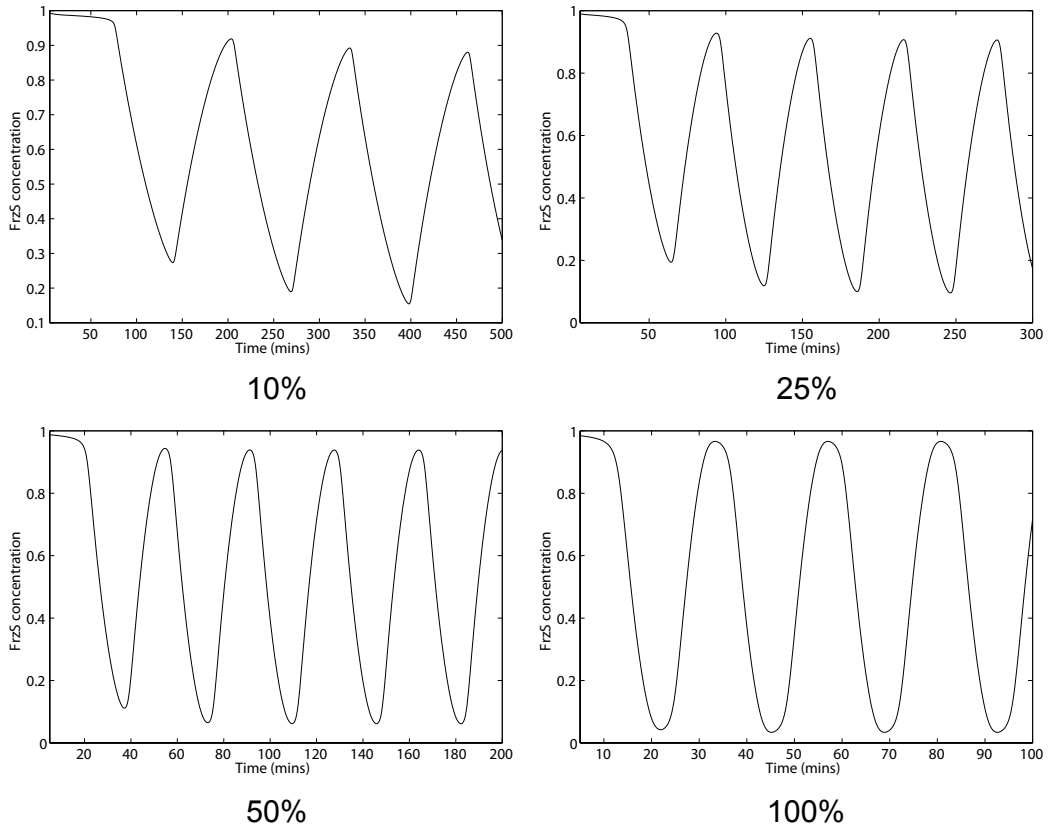


Figure 7.17: The effect of varying MglA on the *Phys-Motilator*. As MglA decreases, the reversal frequency decreases allowing cells to stream. MglA is given as a percentage of the basal value used in the *Phys-Motilator* with 100 % giving a reversal time of 10 min.

Figure 7.17 illustrates the effect varying MglA has on the *Phys-Motilator*. The system has been tuned so that with the value of MglA at 100 % it has a 10 min reversal frequency. As MglA decreases there is a corresponding decrease in the reversal frequency. The *Phys-Motilator* has a continuous response to MglA so that it can be tuned to have a very low reversal frequency giving rise to streaming and fruiting behaviour.

Figure 7.18 shows the reversal characteristics of the cells change during the transition from rippling to streaming. Before the transition stage, nutrients are sufficient and cell reversal frequency is dictated by cell collisions and C-signalling. During and after

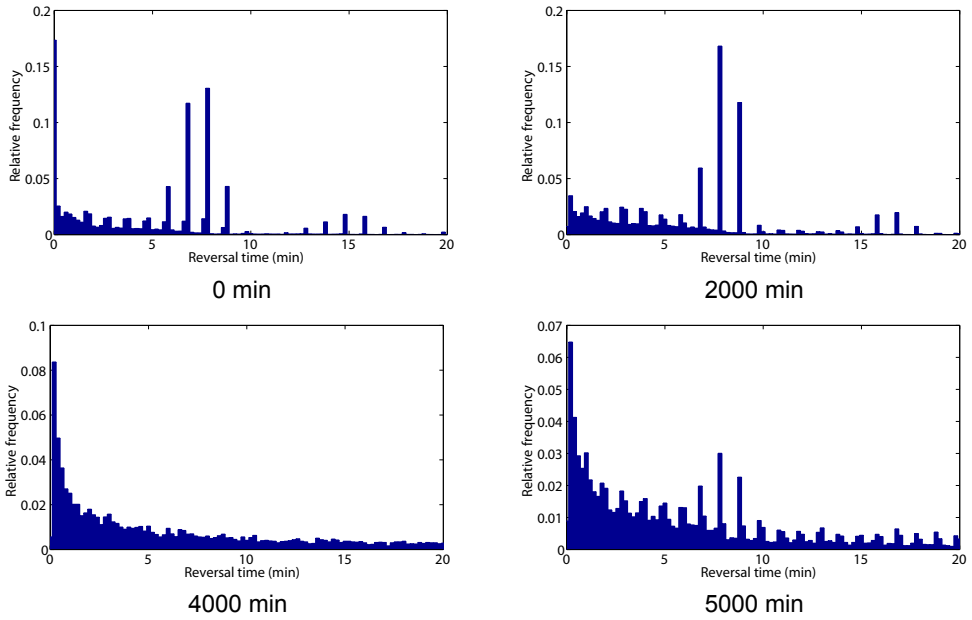


Figure 7.18: Histograms of cell reversal frequencies during the transition from rippling to streaming.

the transition the spread of reversals increases significantly; there is a decoupling effect between C-signal and reversals as cells stop responding to it and move for greater distances between reversals.

Figure 7.19 is a space time plot showing how the *Phys-Motilator* allows cells to transition from rippling to streaming. Importantly it also indicates how cells can go from rippling to streaming. When cells disperse from ripples they are travelling in different directions and subsequently form incongruous streams. Eventually the streams meet and collide and, since cells are no longer reversing, traffic jams form. Two aggregation centres form which rapidly expand as cells move into them (see Figure 7.19(b)). The nutrient source covered the floor of the simulation volume (xy-plane) and was normalised to a value of 1. Each cell consumed nutrients at a rate n_d per time step (see Table 7.1) The system was tuned so the source was depleted after approximately five hours to give sufficient time for ripples to form. When nutrients are maximal, the response is the same as for the constant nutrient model (see Section 7.6.7) and ripples begin to form after 250 min. The ripples grow and become more stable for a further 100 min. After 360 min, nutrients have begun to run out and the ripples begin to dissipate as the cells begin to stream. Although ripples form during the initial half of the

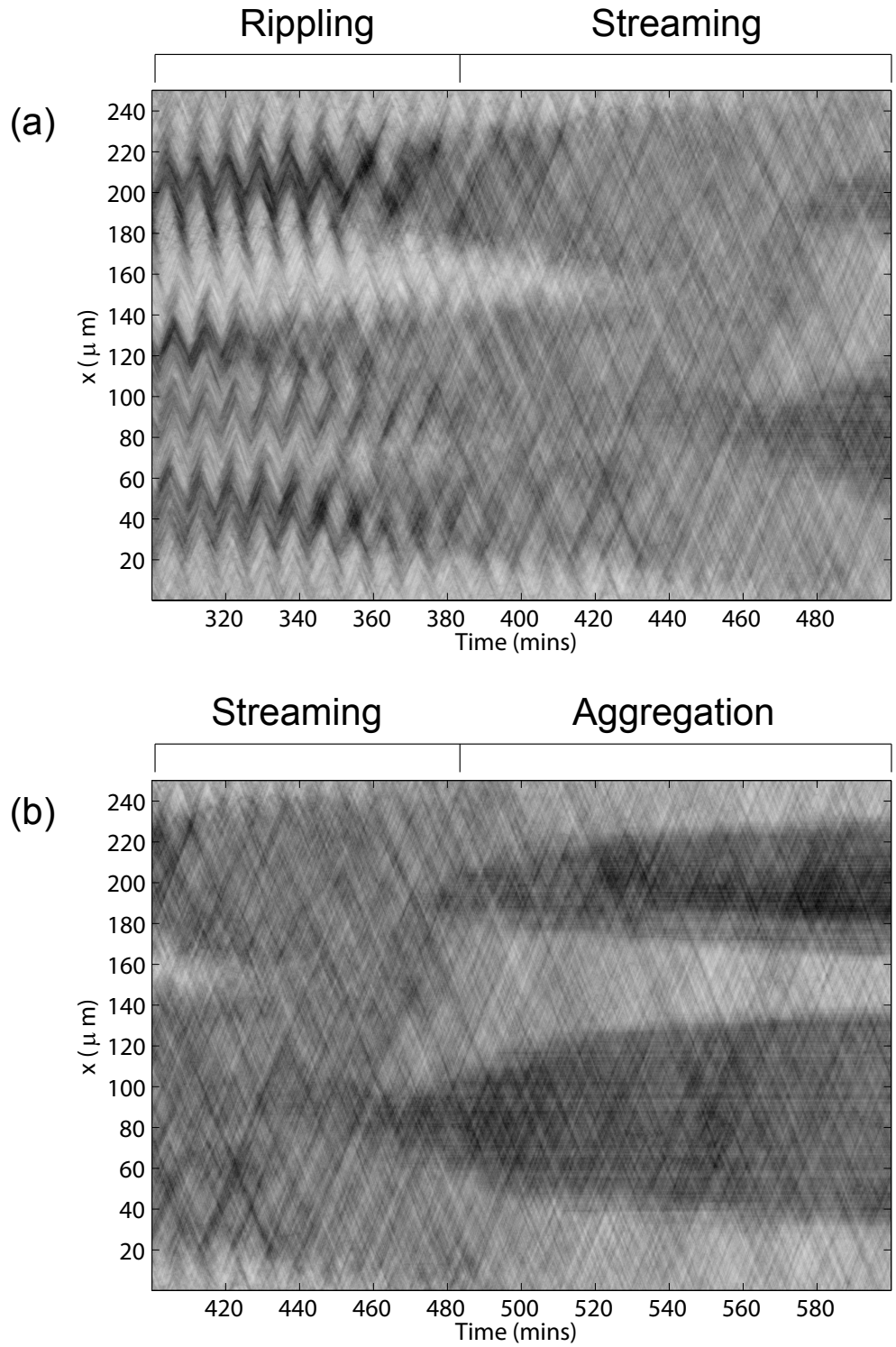


Figure 7.19: Space time plots of 5500 simulated cells showing the transition from rippling to streaming. (a) Ripples have formed by 300 min. After 360 min the ripples break down and the cells migrate away from the ripple zones in streams. (b) After 450 min, large streams of cells have formed, which collide and cause the formation of aggregation centres. The aggregation centres expand outwards as more cells join them.

simulation, they dissipate and eventually dissolve into streams in the absence of nutrients. The *Phys-Motilator* offers a plausible mechanism for controlling cell behaviour and agrees with the evidence of Berleman et al. [10, 11, 12].

7.7 Discussion

Models of myxobacteria motility tend to fall into one of two categories: those concerned with cell biology [62, 63] and those concerned with physical modelling [165, 200, 201]. The *Phys-Motilator* combines both approaches to describe the dynamics of the Frz pathway coupled to a physically realistic model of cell behaviour. Combining both approaches helps validate whether our understanding of both the biology and cell morphology is sufficient to explain spatial phenotypes.

The *Motilator* showed that a model of FrzS protein translocation is sufficient to explain rippling in a simplified CA model of a myxobacteria population. The *Phys-Motilator* extends the *Motilator* so that nutrients also have an effect on the system. The original version of the *Motilator* used C-signal as a triggering mechanism to instigate reversals but it did not address the issue of increased C-signal reducing reversal frequency [70]. Given current knowledge of the Frz pathway, it is probably an as yet uncharacterised external system that regulates C-signal sensitivity. Using recent evidence of rippling being a consequence of cells using localised nutrients [11], FrzG seems to be acting as mediator of FrzCD controlling FrzZ and FrzS. Coupling these effects into the *Phys-Motilator* explains how cells can migrate from rippling to streaming using the Frz pathway.

The *Phys-Motilator* integrates an ODE system with a physical model of cell dynamics allowing the ODE component to be investigated to show it is sufficient to explain ripple formation in a physical model of a cell. There are a number of interesting conclusions, which are not obvious from previous CA models of rippling.

Even in a region of high cell density, cells do not appear to experience a significant increase in C-signalling due to the C-signalling mechanism being only at the cell poles. The small surface area of these regions implies that either cells are very sensitive to C-signal if reversals can be triggered after a few collisions or else C-signalling

occurs along the length of the body. Although C-signal accumulates within a cell up until the point of sporulation, the current understanding of the Frz pathway would suggest that it is not the amount of C-signal within the cell but rather the change in C-signal that causes reversal. If the *Motilator* (or *Frzilator*) are continuously stimulated by constantly increasing levels of C-signal, cells will continue to reverse more quickly leading to hyper-reversals. In reality this does not happen; reversal frequencies return to their pre-rippling values as cells become synchronised within a ripple. This suggests cells only signal during head-head contact otherwise rippling cells would signal continuously and the ripple wavelength would be very short due to much higher reversals. C-signal interaction with the Frz pathway is not straightforward. If the C-signalling effect is finite, then how long a pulse lasts becomes important. CA models typically resolve time in the order of minutes with each iteration representing one or more minutes of time. In the *Phys-Motilator*, time is resolved in the order of seconds so a pulse lasting one iteration is much shorter and consequently has minimal impact on the response of FrzS. It was found that C-signalling must last several seconds if it is to have any impact on the reversal characteristics of the cells.

Within a CA model, cells typically organise along a small, finite number of orientations where cell flexibility is not considered but it does have an effect on the *Phys-Motilator*. As ripples form, areas of low cell density develop. Due to the stochastic motion of cells, small perturbations on cells in a low density area will often lead them to gradually but significantly alter their course. Slime serves an essential purpose of maintaining long term cohesion between cells. S-motility is too short range; cells will remain aligned within small clusters but the macroscopic behaviour allows cells to change course leading to the same dispersal problem as with single cells in a low density environment. Ripple formation is dependent on cells remaining aligned to maximise collisions in counter-propagating waves and troughs. If cells drift off course the ripple fronts break down.

Alignment energy models the effects of S-motility and is dependent on the general direction of movement of a cell's neighbours. Cells tend to align along the average direction of neighbouring cells. If the angle between the average direction of a cell and

its neighbours is obtuse, a cell will align in the opposite direction so that the alignment is direction independent. It was found that if cells only align in the direction of their neighbours, it prevents them from intermixing during collisions and when they are tightly packed. Cells stall and attempt to fold back in on themselves to align with the oncoming cells rather than attempting to push past each other.

The *Phys-Motilator* handles collisions within the Hamiltonian to prevent the system becoming too prescriptive. Wu et al. [200], for example, use an algorithm for collision resolution so cells will always behave in the same manner (with a small amount of variability). It seems unlikely that cells have a mechanism for collision resolution; their movement is a more random process where their behaviour makes them unlikely to experience collisions. If and when they do collide, they attempt to move with their normal behaviour patterns until they are free rather than using a collision resolution strategy. To model this, a collision term was added to the Hamiltonian to make it energetically unfavourable for a cell to occupy the space of another. The centres of each segment must never overlap so there is a large energy penalty for doing so but the surrounding volumes are allowed to overlap to simulate cell compressibility. The natural evolution of the system avoids collisions and uses a random behaviour.

Predation on nutrients and its regulation of FrzCD seem to offer a promising mechanism for explaining the transition from rippling to streaming without needing to further complicate the Frz pathway with additional putative systems to control C-signal. The physical translocation of FrzS does not in itself explain how cells move from rippling to streaming. Rather it seems likely that there are two separate processes involved: the mechanism regulating C-signal levels (for example FrzF and FrzG) and the response mechanism (the *Motilator*). This offers a separation of function rather than combining all the functionality into a large complex monolithic system. Since there is evidence to show FrzG can affect ripple wavelength (and suppress them) it seems more likely that it is existing components of the Frz pathway that control response to C-signal rather than an as yet undiscovered mechanism further along the pathway.

The formation of ripples is the result of cells undergoing many premature reversals until their reversal cycle matches that of their neighbours. Less than 10 % of cells are

reversing prematurely at a given time so ripples form gradually. A consequence of a minority of cells experiencing high levels of C-signalling is the appearance of fewer ripples which are more dispersed. Individual ripples typically form approximately one wavelength apart so that counter propagating waves do not collide. Figure 7.12 shows the formation of ripples in the off lattice model and correspondingly Figure 7.13 shows a space time plot of the simulation.

Cells require some form of collision avoidance in order to ripple otherwise they frequently stall and form traffic jams. Collision occurs in the model whenever the segment of one cell overlaps the volume occupied by another cell. Cells are prevented from doing this and stall until space becomes available. As a consequence, at high cell densities the movement of individual cells is reduced. In reality, cells do glide over each other. Börner et al. [20, 21] allow cells to burrow into opposing streams; however, this effectively prevents cells from stalling since they can always push others out of the way. Given the size and shape of each cell plus the effect of fibrils and pili, it is questionable whether cells would always be able to push through multiple layers of cells. The solution presented here of allowing cells to climb when they are tightly packed appears to be sufficient to alleviate the problem of stalling and makes fewer assumptions about the strength of cells; it relies on a simple pushing effect that drives cells up and over others.

The *Phys-Motilator* can explain rippling and the transition from rippling to streaming provided cells possess certain key physical characteristics. It is the effect of all these parameters combined that allows ripples to form. Rippling cannot be explained purely by the function of the Frz pathway alone. The combined effect of FrzG and FrzF seems to be plausible.

Chapter 8

Fruiting body morphogenesis

The final results chapter looks at fruiting body morphogenesis showing how the unified model of C-signalling and cell motility developed in Chapter 6 and Chapter 7 can also explain multicellular aggregation. Together with Chapter 6 and Chapter 7 it is hoped that this body of work presents a more comprehensive understanding of the myxobacterial life-cycle and sets the groundwork for future models that refine what is presented here.

8.1 Introduction

Bacteria are able to sense their surroundings in order to adapt to environmental change. Most bacteria live in dense populations, therefore other cells constitute a major part of their physical and chemical stimuli allowing regulatory interactions between cells to be established. The benefits of coordinated behaviour include: more efficient proliferation resulting from a cellular division of labour, access to resources that cannot be utilised by isolated cells, defence against antagonists and population survival by differentiation into distinct cell types [146].

The myxobacteria developmental process (details of the biology of fruiting body development are given in Section 2.7.4) involves a series of macroscopic changes in colony morphology. Figure 8.1 shows the four primary stages of fruiting body deve-

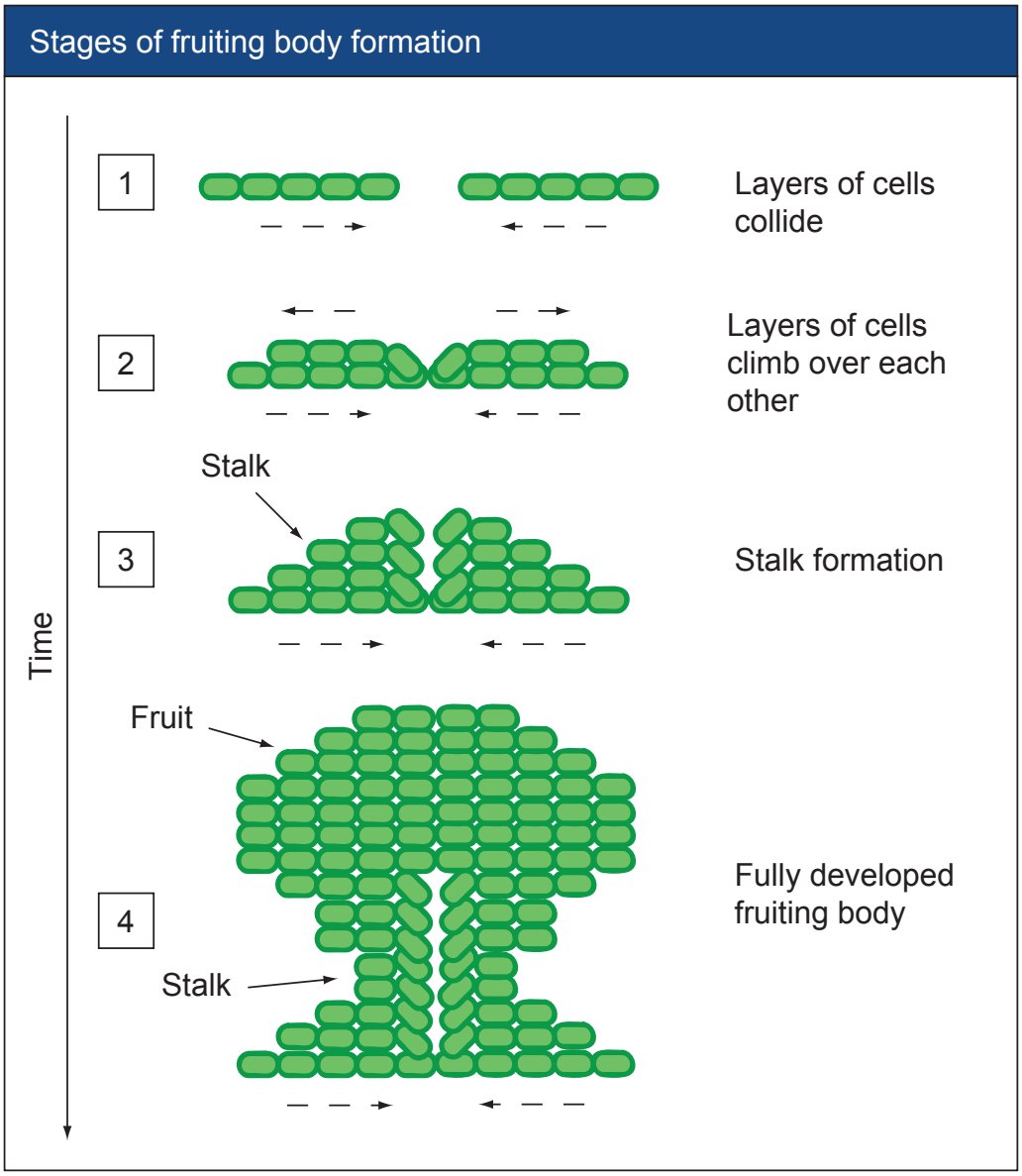


Figure 8.1: The main stages of fruiting body formation. Initially streams (single sheets of cells) will come together and collide. One of the sheets moves on top of the other building a second layer of cells. Layers continue to build forming a stalk. The structure continues to expand upward and outward forming the fruit on top. As the stalk continues to build, the fruit is pushed upwards.

lopment. A key regulator of development is C-signalling which occurs when C-signal, a cell surface-associated signal encoded by *csgA*, is exchanged between cells in contact with one another. C-signal stimulates the expression of *csgA* leading to positive feedback and a rise in C-signalling throughout development. Different colony morphologies are a consequence of different C-signalling levels [86]. C-signalling is thought to affect the reversal frequency of individual cells in a contact-dependent fashion allowing the synchronisation of behaviour [84, 86, 89].

During vegetative growth cells move in the direction of their long axis, reversing typically once every ten minutes [71]. Under starvation conditions, C-signal accumulates within a cell [86] reducing its reversal frequency [70]. The lack of reversal and the effects of A and S motility causes cells to form streams and increases the likelihood of aggregation; cells which cannot reverse tend to remain stuck in one location since their ability to move around obstacles is limited by only being able to move forward [202].

Myxobacteria begin to form fruiting bodies after a prolonged starvation period of approximately 24 h. Starved cells form into large, intricate multicellular aggregates containing between 50,000 and 100,000 myxobacteria cells [118]. The fruiting body is the precursor to sporulation where cells undergo morphogenesis and physically change shape from rods to nearly spherical cells (although there is some degree of variation and not all cells undergo such drastic transformations) [189]. Inside the nascent fruiting body, a percentage of the cells differentiate into dormant myxospores. This process requires both temporal and spatial coordination in three-dimensions making it one of the most complex and least understood phases of the life-cycle. The majority of the research on the dynamics of fruiting was carried out over 20 years ago [118, 136, 182, 183] and relatively little is known about the spatial dynamics of fruiting body construction with research primarily devoted to understanding the signalling mechanisms required to coordinate development rather than cell physics [31].

During starvation, C-signal accumulates within a cell [86] reducing its reversal frequency [70]. By the time cells have reached the fruiting body stage, they are reversing very infrequently. The lack of reversal and the effects of A and S motility causes cells to form streams and increases the likelihood of aggregation; cells collide and tend to

remain stuck in one location since their ability to moving around obstacles is limited by only being able to move forward [202].

There is some disagreement over how fruiting actually begins. O'Connor and Zusman [118, 190] observed that cells appear to orbit around a largely stationary aggregation centre. This led to the *traffic jam* model, which proposes that streams of cells collide together causing the formation of a kernel of stationary cells around. Cells move around and over the static centre leading to a mound formation aggregation [65]. Work on *Stigmatella* showed that cells form circular orbits around a base and then move upwards in a spiral fashion around the base, building the stalk on top of it [50, 190]. It was presumed all myxobacteria form fruits in a similar way; however, Kuner and Kaiser [90] did not observe the spiralling patterns suggesting that this behaviour is possibly non-essential and may not be intrinsically important to fruiting development. Recent work by Curtis et al. [31] suggests that myxobacteria actually form fruits using a stepped layer building approach:

- Streams of cells form moving sheets.
- Sheets eventually collide causing a rapid build up in density at the meeting point.
- Cells in one of the opposing streams cells are forced upwards and over the other, similar to tectonic plate movements.
- The displaced cells are supported on top of the highly dense layer of cells and EPS slime underneath and begin to spread out forming a new layer.
- As the new layer becomes more dense itself, cells at the centre start to get pushed upwards to form a new layer and the process repeats causing the formation of an expanding mound of cells.

Previous computational models of fruiting body development [81, 160, 161] are based on the *traffic jam* model and rely upon the artificial induction of an aggregation centre, typically by making a subset of cells stationary, even though there appears to be limited evidence to show that cells construct fruits in this manner.

In this chapter an off-lattice Monte Carlo simulation based on the observations of Curtis et al. [31] is used to show how cells can spontaneously aggregate to form layers and fruiting bodies. The motivation of the work was to use mathematical and computational modelling to study the important physical characteristics cells must possess to engage in fruiting. The *Phys-Motilator* is designed to cope with changing levels of MglA which is in turn affected by C-signal and nutritional stimulus. As discussed in Section 6.7.2, the *Phys-Motilator* can be tuned to have a very low reversal frequency (see Figure 6.8). This is implemented within the fruiting models to capture the effect of very high levels of C-signal. A Monte Carlo approach is adopted since it allows the *Phys-Motilator* to be extended to cope with fruiting. Stochastic models have been used successfully to study three-dimensional fruiting formation in *Dictyostelium discoideum* [142].

8.2 Modelling fruiting body formations

8.2.1 Myxobacterial cell design

Simulations were carried out in three-dimensional environment using a model based on a previously developed stochastic model of myxobacteria motility [60] and the model presented in Chapter 7. This extended model will be referred to as the *Fruit-Motilator*. *M. xanthus* is approximately 5 μm to 7 μm long and 0.5 μm in diameter so the model cells were given a length to width ratio of 7:1. Each cell was composed of eight three-dimensional cuboidal segments (see Figure 8.2) with each segment being composed of 27 segment nodes arranged in a cube formation. A small overlap controlled by the stretching energy term of the Hamiltonian (see Section 7.5.8) was allowed so that cells maintain a continuous volume despite being made of multiple separate segments (see Figure 8.2(c)). The physical behaviour of cells was described using a Hamiltonian function whilst the internal state is described using ordinary differential equations (ODEs).

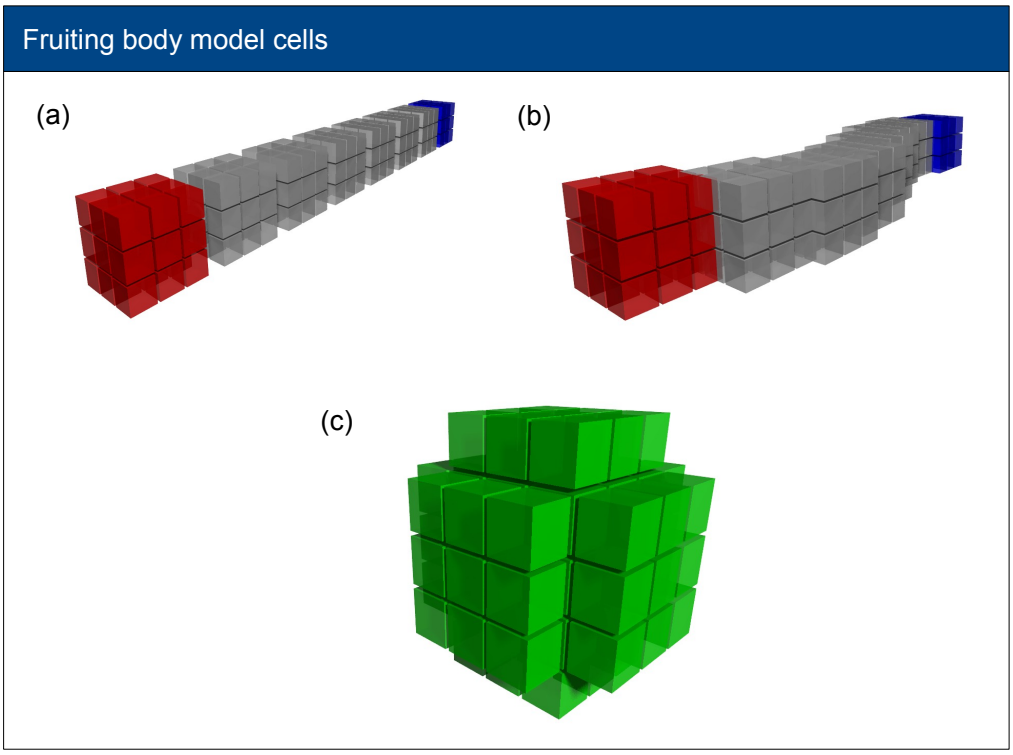


Figure 8.2: Cell design for fruiting body simulations. Each cell is composed of a number of connected segments with each segment being composed of a number of nodes. (a) Fruiting body cells have eight segments: a head (red), a tail (blue) and six body segments. Each segment comprises 27 segment nodes in a cube formation. (b) Segments move independently allowing the cell body to be flexible. Overlap between segments allows the cell to maintain a continuous cell volume. (c) During sporulation cells change shape becoming immobile single segment cells. A spore is constructed as a three-dimensional sphere of segment nodes. The physical characteristics of each cell is described using a Hamiltonian function.

8.2.2 Simulation volume

The shape and size of fruiting bodies varies greatly; however, a typical mature fruit consists of a short stalk and mound formation (see the fruiting body formation in Figure 2.5 at the 61 h stage) with a diameter of approximately 100 μm [31, 50, 81, 90]. A simulation volume equivalent to 60 μm$\times$ 60 μm$\times$ 30 μmwas used for all simulations to allow fruiting bodies to develop. The work presented here is concerned with the initial formation of the fruit so a large simulation volume to contain a mature fruit was unnecessary.

Periodic boundaries are implemented except in the xy-plane since it does not make sense for cells to be able to push through the floor nor move through the ceiling since that would imply burrowing through the floor of the next domain and under the exis-

ting cells for which there is no physical interpretation. Boundaries are maintained with a boundary energy term which severely penalises a cell for attempting to cross a particular domain boundary. The energy penalty is several orders of magnitude larger than the value any of the other energy terms might produce (positive or negative) so it is impossible for a configuration with a domain crossing to be favourable.

8.2.3 Cell influx

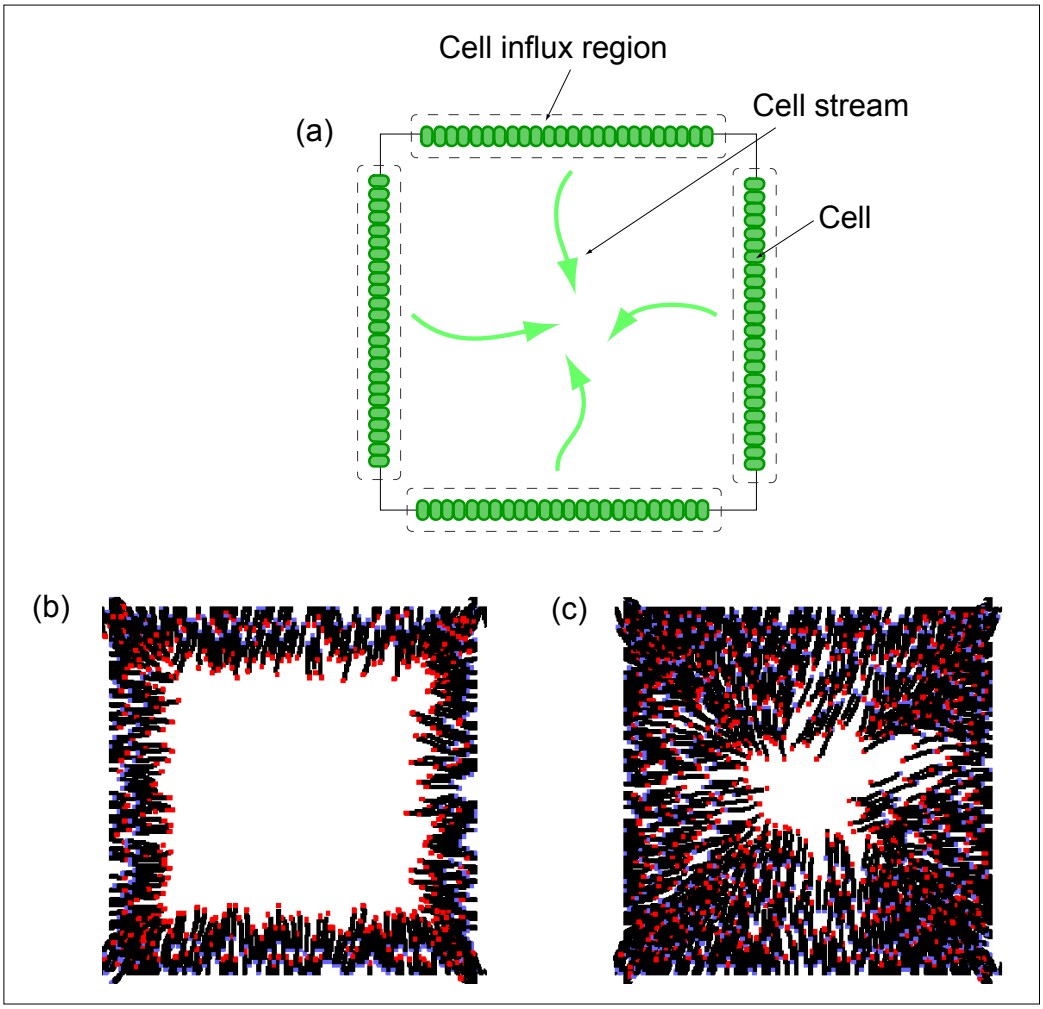


Figure 8.3: Converging stream formations. Fruiting body simulations begin with streams of cells converging to form an aggregate. To maintain cell density within the fruit, the initial stream formations are augmented with four cell influx regions, one at each boundary of the simulation (in the xy-plane). Cells are created at the influx regions and allowed to move into the simulation volume. (a) Diagram view of simulation. (b) Snapshot of a simulation after 20 time steps showing the creation of cells. (c) Snapshot of the same simulation after 50 time steps showing the formation of streams of cells converging towards the centre of the simulation. Cells move in the direction of their head (red segments) from their tail (blue segments).

Fruiting body formation requires a highly dense region of cells to seed aggregation. To achieve a high cell density within the simulations, a virtual stream model was devised (see Figure 8.3). *birth zones* were placed around the edge of the simulation volume, which introduce new cells into the environment. Cells are rapidly born in the birth zone as small two segment entities which rapidly grow into full size cells within a few time steps. This does not represent a biological phenomenon, but rather is to ensure that as the stream moves forwards it is constantly being replenished by new cells with the effect of making the streams infinitely long. Fruiting requires a high cell density and simulating a finite number of cells makes it difficult to assemble enough cells in an area to form a fruit. It is not that the underlying rules of the model are wrong, but the cell density is never high enough; a small, finite number of cells may clump and aggregate but there will never be enough to cause a fruit to form. With the birth method, a constant cell density can be maintained to keep providing cells to form the fruit.

8.3 Algorithm implementation

Algorithm 2 is an overview of how the model is implemented. Simulations were carried out using the parameters listed in Table 8.1. It should be noted that unlike Wu et al. [200], a separate collision resolution algorithm was not required since collision avoidance is a feature of the Hamiltonian. The model was implemented using FABCell as described in Section 6.5

8.4 System Hamiltonian

The Hamiltonian for the fruiting model is based on the one used by the *Phys-Motilator* (see Chapter 7). It maintains the key features of the *Phys-Motilator* and includes additional terms describing important physical characteristics of fruiting cells (summarised in Table 8.2). Reversal frequency is very low during the fruiting period. The *Fruit-Motilator* was tuned to give a very long reversal time (see Figure 6.8). The following Hamiltonian function describes the energy of the system:

Algorithm 2 Fruiting development

for $c = 1$ to C (number of cells) **do**
  for $i = 1$ to F (number of Monte Carlo flips) **do**
    Determine cell interactions with neighbouring cells.
    Update the internal state of the cell.
    Propose a new location for the head node to move to. Each new location is a fixed distance L from the current head position.
    Apply the Metropolis algorithm [106] to determine the acceptance probability of making the change.

$$P(\Delta E) = \begin{cases} 1 & \text{if } \Delta E \leq 0, \\ e^{-\Delta E/kT} & \text{if } \Delta E > 0 \end{cases} \tag{8.1}$$

    if move is energetically favourable **then**
      Move the head
    else
      for $j = 1$ to F **do**
        Repeat steps 3 to 6.
        if new location found **then**
          Move the head.
          Exit for loop.
        end if
      end for
      for $j = 1$ to $F \times N$ **do**
        Choose segment k at random from cell c.
        Propose moving segment in the direction from segment k to segment $k + 1$ (the head segment of k) a distance L.
        Apply the Metropolis algorithm.
        if move is energetically favourable **then**
          Move the segment.
          Write out the slime vector at the location of the tail.

$$S = \frac{s_{a,1} - s_{a,N}}{\|s_{a,1} - s_{a,N}\|} \tag{8.2}$$

        end if
      end for
    end if
  end for
  for each cell less than N segments in length **do**
    Add a new segment to the tail of the cell.
  end for
end for
for each cell birth region **do**
  for each empty location **do**
    Create a new cell one segment in size and place it at the location.
  end for
end for

Table 8.1: Parameters for models used to simulate fruiting body formation in *Myxococcus xanthus*.

Name	Value	Description
λ	3.0	Dimensionless stretching energy parameter.
α	2.0	Dimensionless volume energy parameter.
σ	12.0	Dimensionless bending energy parameter.
ϵ	2.0	Dimensionless propulsion energy parameter.
ν	1.0	Dimensionless climbing energy parameter.
μ	3.0	Dimensionless gravity energy parameter.
τ	100.0	Dimensionless collision energy parameter.
kT	0.3	Boltzmann constant $\times$ temperature.
n_s	8	Number of segments per cell.
V_s	0.422 μm^3	Segment volume.
d_0	0.75 μm	Target distance between adjacent segments ($\sqrt{V_s}$).
C_s	50	C-signal sporulation threshold.
P_s	0.1	Probability a cell will sporulate.
V	$60 \times 60 \times 30$ μm^3	Simulation volume.

Table 8.2: The role of specific energy terms in the fruiting body Hamiltonian. Terms are in addition to, or replace, those described in Table 7.2.

Term	Description
$E_{adhesion}$	A fruiting body contains cells at a very high density. The cells produce a polysaccharide slime which encases the cells in a slime matrix and affects the adhesion between cells [50, 153].
$E_{climbing}$	A more sophisticated version of the climbing function presented in Table 7.2 is required to allow cells to climb on and around the fruiting body.
$E_{gravity}$	Gravity prevents cells from climbing excessively or in empty space.

$$\begin{aligned}\mathcal{H} = E_{adhesion} + E_{align} + E_{bend} \\ + E_{climbing} + E_{collision} + E_{gravity} \\ + E_{propulsion} + E_{slime} + E_{stretch}\end{aligned} \tag{8.3}$$

The following terms are specific to the fruiting body Hamiltonian function. Terms not specified are the same as those described in Section 7.5.

8.4.1 Adhesion energy

In a highly dense region, cells generate a lot of polysaccharide slime with a fruit being a large amalgamation of cells and extra-cellular matrix. The slime probably has a surface tension effect causing cells to stick together rather than drifting apart. This is different to the slime trail following as it is non directional, acting over the whole cell area. If two cells are close to each other and encased in slime then to break apart requires extra energy to counter the adhesive effects of the slime. In contrast to the climbing effect, here cells experience an energy penalty for breaking apart. It is a form of non-specific attraction and operates over short ranges since two cells several cell lengths apart will not be affecting each other, only cells in close proximity will experience adhesion.

The high density of cells in a swarm and fruiting body means there is a large amount of polysaccharide slime produced which encases all of the cells in a slime matrix [50, 118, 153]. The slime casing prevents cells coming apart, for example even with a rotary shaker. This matrix effects an adhesive force on the cells making it harder for cells to move apart from each other. Cells typically aggregate at a colony edge due to surface tension effects making it difficult to escape the colony [81].

This effect is different from the effects of A-motility and is a global property of a large mass of cells.

$$E_{adhesion}(a) = -\varphi \sum_{i=2}^{N} \sum_{(j,k) \text{ neighbours}, j \neq a} \frac{1}{\|\boldsymbol{s}_{a,i} - \boldsymbol{s}_{j,k}\|^2} \tag{8.4}$$

where φ is a dimensionless adhesion coefficient and $\boldsymbol{s}_{a,b}$ is the coordinate of the centre of segment b of cell a.

8.4.2 Climbing energy

Curtis et al. [31] observe that cells appear to move in sheets towards each other and upon impact during a collision, cells from one sheet can move up and on top of the other. This is consistent with O'Connor and Zusman [118] who suggest that cells appear to behave as independent sheets. This effect has been modelled so that it is somewhat analogous to a snow plow. The plow is pushed forwards into the snow pushing the snow up and away. In a similar way it is envisaged that the oncoming force of a sheet of cells is sufficient to push opposing cells up and direct them over the top. A cell monitors the number of head on collisions it is having with the more the cells the greater the chance of being pushed up. Cells are not forced to be always pushed upwards since this would be imposing an artificial constraint on the system, instead cells prefer regions of lower cell density. Some cells will be pushed outwards away from the stream, but the majority will be pushed upwards since this is generally the region with the most free space available.

Climbing energy was considered in the *Phys-Motilator* model (described in Section 7.5.3); however, a modified version is presented here for the fruiting model. Curtis et al. [31] propose that when two sheets of oncoming cells encounter each other, individual cells have a proclivity to move out of the potential "traffic jam" that can ensue and typically this is upwards so one sheet of cells effectively moves over the other. A simulation of climbing cells which form layers can be seen in Figure 8.4. The energy term described here encourages cells to move upwards proportional to the number of oncoming cells they are interacting with.

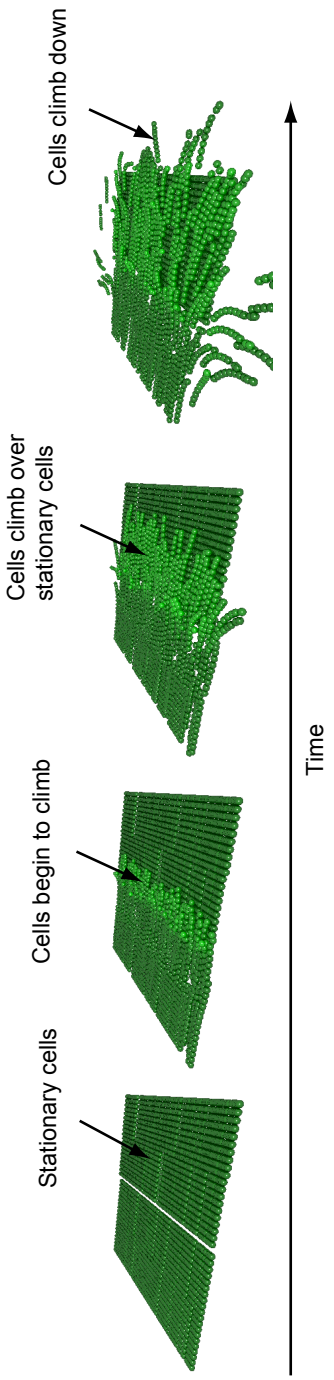


Figure 8.4: A model of cell climbing and layer formation. When cells encounter an obstacle such as stationary cells they either attempt to move around it or climb over it.

$$E_{climbing}(a) = -\eta \times \text{cells}(a) \times \text{dir}(a) \tag{8.5}$$

$$\text{cells}(a) = \sum_{i \text{ neighbours}} \text{collision}(a, i) \tag{8.6}$$

$$\text{collision}(a, i) = \begin{cases} 1 & \text{if } \hat{\boldsymbol{d}}_a \cdot \hat{\boldsymbol{d}}_i \leq 0, \\ 0 & \text{else.} \end{cases} \tag{8.7}$$

$$\hat{\boldsymbol{d}}_m = \frac{\boldsymbol{s}_{m,1} - \boldsymbol{s}_{m,N}}{\|\boldsymbol{s}_{m,1} - \boldsymbol{s}_{m,N}\|} \tag{8.8}$$

$$\text{dir}(a) = \hat{\boldsymbol{r}}_a \cdot \hat{\boldsymbol{n}}, \tag{8.9}$$

where η is a dimensionless climbing coefficient, cells(a) determines the number of oncoming cells, collision(a, i) determines if two cells are going in opposing directions by examining the dot product between the normalised average direction ($\hat{\boldsymbol{d}}_m$) of each pair of interacting cells and dir(a) compare the direction cell a wants to move in ($\hat{\boldsymbol{r}}_a$) with a normal vector (typically a normal to the xy-plane).

8.4.3 Gravitational energy

Objects on Earth are subject to a gravitational force. In a three-dimensional model, cell movement in the z-axis needs to be controlled; cells cannot randomly climb into empty space and defy gravity. The other energy terms do not prevent cells from climbing because they primarily maintain cell shape and ensure cell motion is along reasonably straight paths, which can be in any dimension; vertical ascent is energetically indistinguishable from motion along the ground. Gravity is therefore introduced as an energy penalty for trying to climb; the steeper the climb the greater the penalty. An object acting under gravity requires the greatest amount of energy to directly oppose the force and move in the opposite direction. The gravitational term rewards a cell for moving downwards. It should be noted that the use of the dot product means that there is no net effect of this term for a cell moving in a straight line in the xy-plane. Since gravity is a constant, there is no change in energy from moving between two positions with a direction vector perpendicular to the direction of the force.

$$E_{gravity}(a) = -\mu \left(\left[\hat{\boldsymbol{b}}_{a,1} \cdot \hat{\boldsymbol{c}} \right] \cdot \text{space}(\boldsymbol{d}_{a,1}, n) \right) \tag{8.10}$$

$$\hat{\boldsymbol{d}}_{m,n} = \boldsymbol{s}_{m,n} - \boldsymbol{e} \tag{8.11}$$

where μ is a sensitivity parameter, $\hat{\boldsymbol{b}}_{a,1}$ is the normalised update direction of the head segment, $\hat{\boldsymbol{c}}$ a normalised direction vector pointing towards the ground, $\boldsymbol{d}_{m,n}$ is a location below the centre of segment n of cell m and n is a local neighbourhood surrounding $d_{m,n}$.

8.5 A model of sporulation

As a fruiting body matures, 65 % to 90 % of cells will lyse with the remaining cells going on to form myxospores [82, 192]. Spores appear to migrate to the centre of the fruiting body with motile cells remaining on the outside and the periphery [118]

The *Fruit-Motilator* was extended to investigate sporulation and how this affects fruit formation. The model (referred to as the *Spore-Motilator* hereafter) is principally the same as the *Fruit-Motilator*; however, each cell is now given a type: *motile* or *spore*. Motile cells accumulate C-signal from collisions with other motile cells. Once C-signal exceeds a threshold (C_s), cells convert to non-motile spores. Spores can be moved by motile cells pushing them. Each cell type has its own Hamiltonian governing its behaviour. Normal cells continue to use the Hamiltonian defined in Equation 8.3:

$$\begin{aligned} \mathcal{H}_n = {} & E_{stretch} + E_{align} + E_{bend} + E_{propulsion} \\ & + E_{slime} + E_{climbing} + E_{gravity} + E_{collision} \end{aligned} \tag{8.12}$$

Spores are non-motile cells with a fixed size and shape. The Hamiltonian controlling them loses terms associated with autonomous cell motion:

$$\mathcal{H}_s = E_{spore} + E_{climbing} + E_{gravity} + E_{collision} \tag{8.13}$$

Although spore cells are immobile, other motile cells can move them during collisions when they collide and through adhesive effects between cells.

$$E_{spore}(a) = \xi \cdot \hat{\boldsymbol{c}} \cdot \hat{\boldsymbol{e}} \tag{8.14}$$

$$\hat{\boldsymbol{e}} = \frac{\boldsymbol{e}}{\|\boldsymbol{e}\|} \tag{8.15}$$

$$\boldsymbol{e} = \sum_{i \text{ neighbours}} \boldsymbol{s_{i,1}} - \boldsymbol{s_{i,N}} \tag{8.16}$$

where ξ is a dimensionless coefficient, $\hat{\boldsymbol{c}}$ is the normalised average direction of the cell, $\hat{\boldsymbol{e}}$ is the average direction of all the cells in a local neighbourhood surrounding cell a. b_m reflects that cells tend to turn through the acute angle to align with other cells in either direction.

8.6 Results

8.6.1 Model of fruiting body formation

The shape and size of fruiting bodies varies greatly; however, a typical mature fruit consisting of a short stalk and mound formation (see the fruiting body formation in Figure 1 in [90] at the 61 h stage) has a diameter of approximately 100 μm [31, 50, 81, 90]. A simulation volume equivalent to 60 μm $\times$ 60 μm $\times$ 30 μm was used for all simulations to allow fruiting bodies to develop. The work presented here is concerned with the initial formation of the fruit so a large simulation volume to contain a mature fruit was unnecessary.

8.6.2 Fruiting with a finite number of cells

The initial stream formation and adhesion models used a finite number of cells (in reality there are hundreds of thousands of cells within an area of fruiting colony). It is cell density that is required for fruiting formation. In a conventional model with a finite number of cells, it is difficult to achieve a high enough cell density to maintain fruit formation. Fruiting body formation can start from a mono-layer of cells. A model

started with a finite number of cells in a mono layer, even if they occupy the whole environment floor is unlikely to be enough to form a fruit. If the density is too high, cells will be unable to move necessitating an upper limit on the cell density. Once the fruit starts to form, as the layers build up, the influx of cells into the aggregate will not be sustainable and the fruit will simply break down. Although it would be ideal to model a vast mono layer of cells to ensure there are enough cells, computational (typically memory) limitations enforce limitations on the size of a simulation.

Figure 8.5 shows the output of a fruiting body simulation using a finite number of cells. 1600 cells were arranged into two opposing sets of streams with one set perpendicular to the other. The streams move into each other and collide. In the aggregation centre, some cells push upwards and move over others forming new layers and the base of a stalk. The effect of using a finite number of cells becomes apparent after 300 time steps when the stalk begins to disassociate. The cells organise themselves into a stack four layers deep. However, there are no more cells to expand the base layer so the upper cells begin to climb down and move away from the stalk. Once a few cells move away, a mass exodus is triggered causing all of the cells to move away. The formation and subsequent rapid dispersion of fruits will occur at any point where an aggregate forms. This effect will be more apparent on subsequent aggregation formations since the number of cells within the mass is unlikely to be as high as in the initial formation so the deterioration will be more pronounced.

8.6.3 Climbing

In order to escape from collision regions, cells have a limited ability to climb. The climbing energy term in these experiments differs from that in the *Fruit-Motilator* (see Section 7.5.3) in that it considers all possible directions for climbing. The *Fruit-Motilator* climbing term was allowed cells to move vertically up and down or not at all. This was found to be insufficient for studying fruiting. Cells do not climb exclusively vertically since this forces the cell to bend severely and is physically unrealistic. Instead, cells can now climb at any angle, but it is more favourable for them to climb at steeper inclines. Climbing simulates the effect of oncoming cells pushing up and under cells

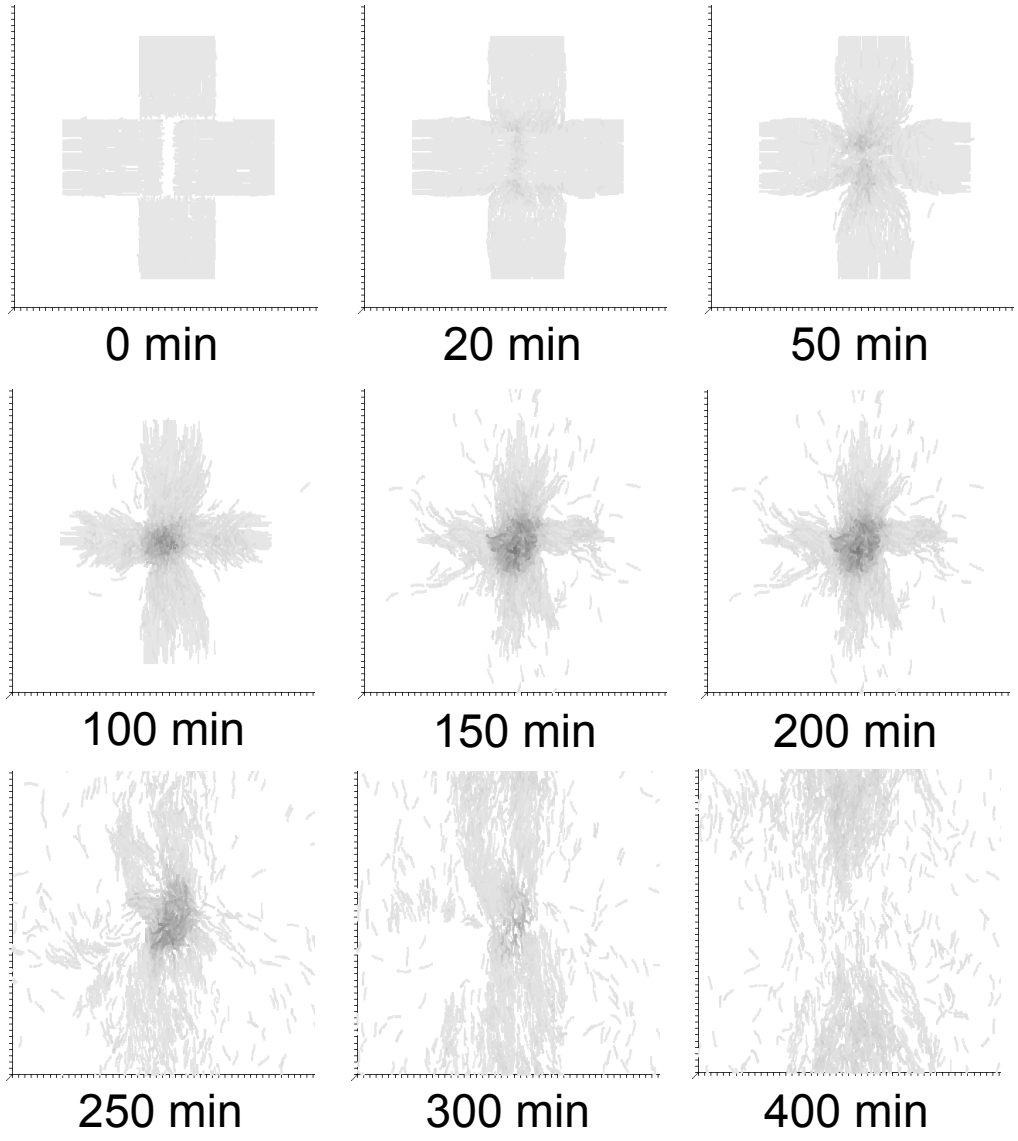


Figure 8.5: Fruiting body formation with a finite number of cells. 1600 cells were divided into four opposing streams. A fruiting body starts to form after 100 min time steps. There are not enough cells to sustain fruiting body growth beyond a few layers and cells dissociate after 400 min. Plots are a two-dimensional (xy-plane) top down view of a three-dimensional environment. Cells are coloured by height; the darker the grey, the higher the cell has climbed.

causing them to rise upwards to a new layer. A typical incline profile is relatively steep ($\gamma > \pi/4$ rad) since a cell is being pushed upwards by another cell so the height it rises must be sufficient to allow the opposing cell to move into its space.

8.6.4 Cell adhesion

The extracellular polysaccharide slime (EPS) that surrounds cells is rarely considered in models, but here it was found to be essential to fruit formation. The Hamiltonian includes an adhesion term, which generates energy proportional to the inverse square of the distance between any two cells in a neighbourhood. It is more energetically favourable for cells to remain close to other cells otherwise there is a severe penalty for moving apart that increases exponentially with distance. An inverse relationship was chosen so that long range interactions are weak; cells towards the perimeter of the local neighbourhood should not exert the same influence as cells in close proximity. Adhesion acts to control the viscosity of the slime determining how easy it is for cells to move through it. The amount of slime and thickness varies depending on the stage of fruiting and the cell density [50].

Figure 8.6 shows the effect of varying the strength of adhesion on a small stream of 300 cells. When there is no adhesion, cells at the front the stream are able to move adventurously causing the stream to break down into a number of smaller streams which diverge. As the adhesion strength (φ) is increased cells remain much closer. When $10 < \varphi < 40$ cells tend to stay as one or possibly two large coherent streams. As $\varphi \geq 50$ it has the effect of making the slime so viscous, cells are no longer able to move.

8.6.5 Aggregate formation

Fruiting begins with streaming and the confluence of streams to form aggregation centres. The fruiting model presented here allows cells to spontaneously form streams and aggregation centres (see Figure 8.7). A simulation consisting of 300 non-reversing cells was run three times to determine the efficacy of streaming and aggregation (model parameters are given in Table 8.1). Cells were initially randomly distributed with a random orientation in a 105 μm $\times$ 105 μm $\times$ 40 μm environment. After approxima-

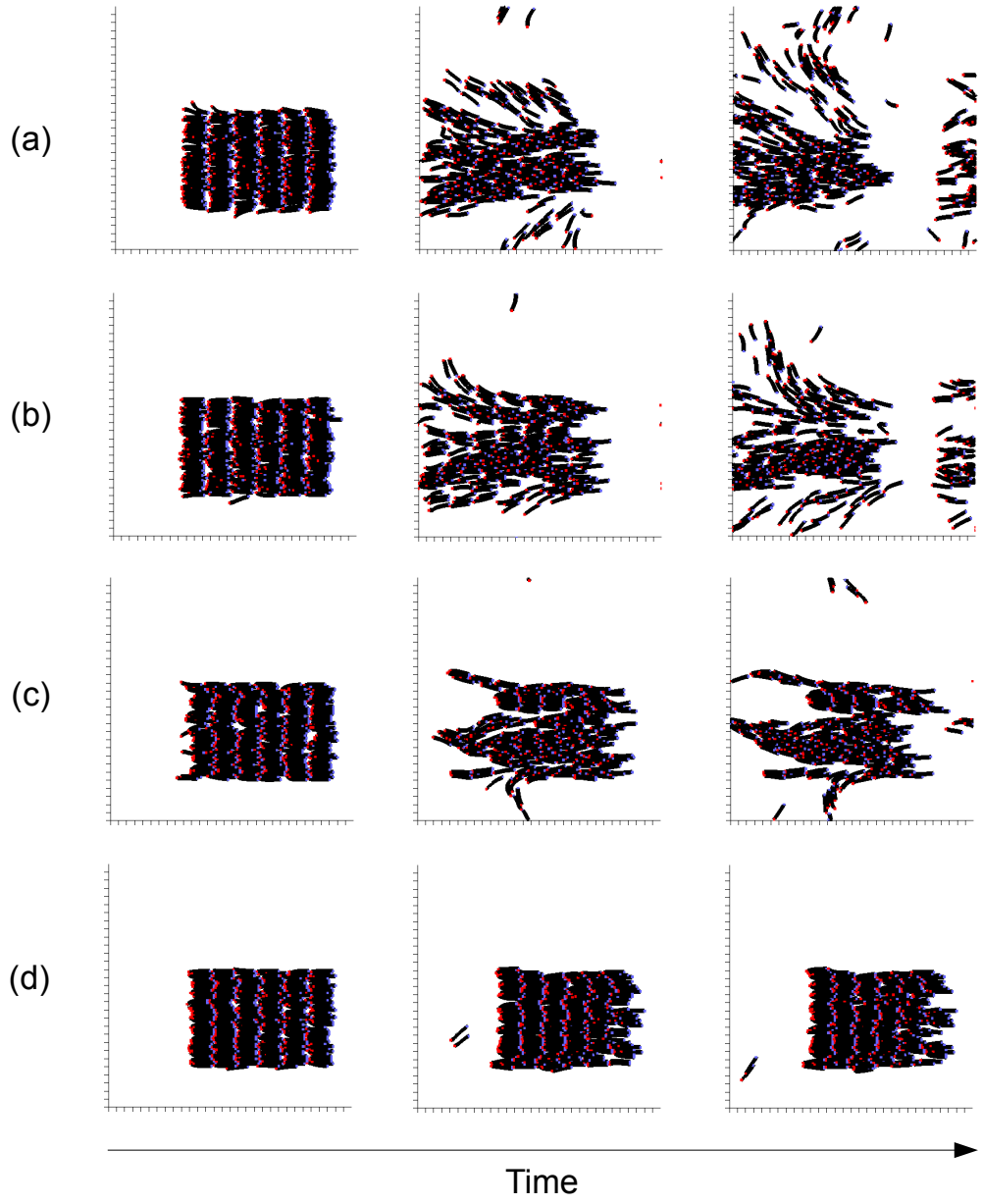


Figure 8.6: The effects of adhesion on stream formation. As adhesion becomes stronger cells cannot break apart and remain together in tighter clusters until the slime effectively becomes so viscous, cells cannot move. The head, tail and body of each cell are coloured red, blue and black respectively. Plots are a two-dimensional(xy-plane) top down view of a three-dimensional environment. (a) $\varphi = 0$. (b) $\varphi = 10$. (c) $\varphi = 40$. (d) $\varphi = 50$.

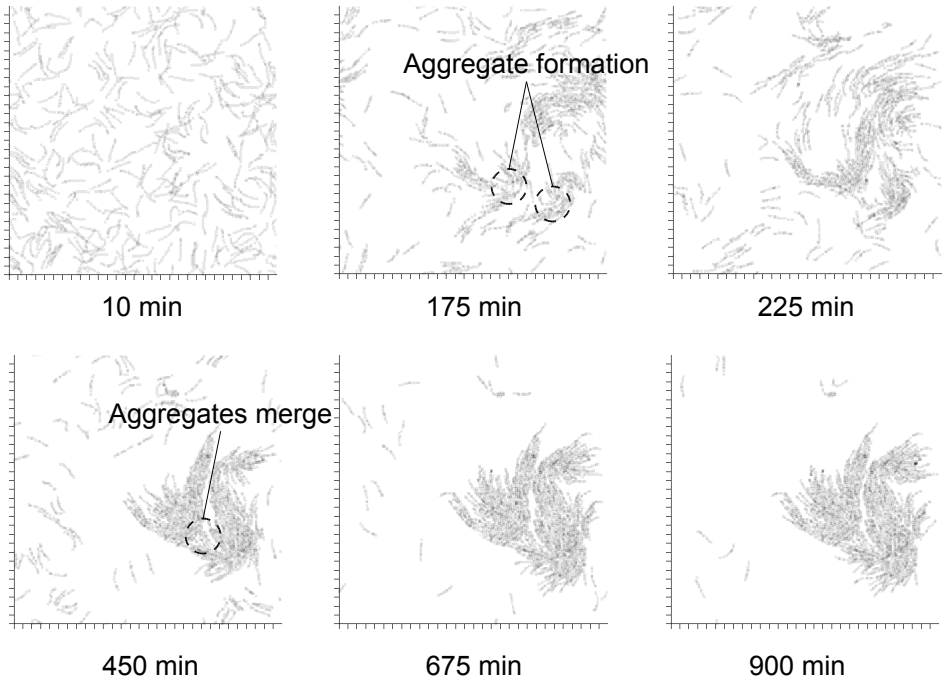


Figure 8.7: Simulation of 300 cells using the fruiting Hamiltonian and variant of the *Motilator* which is non reversing. Two aggregation centres spontaneously form. They eventually merge into one large aggregation centre where a fruiting body can form. In this model a fruit will not form as the cell density is not great enough to sustain building.

tely 100 min of simulate real time, cells formed into streams regardless of their initial configuration. Cells aligned and formed small streams which joined other streams when they came into contact. After300 min cells typically formed an aggregate, which expanded as the the majority of cells joined it.

The effects of motility along with cell adhesion causes model cells to form streams. As the streams approach an aggregation centre, cells will attempt to avoid collision and alter course. They begin to move around the aggregate causing the stream to change direction and form the characteristic spiral patterns observed by O'Connor and Zusman [118] (see Figure 8.16(a)).

8.6.6 Fruit dispersal

Curtis et al. [31] observed that during the initial stages of fruiting, small fruiting bodies would sometimes repeatedly start to form and then dissipate before a stable fruiting body finally formed (see Figure 8.8). The formation of transitory aggregates can be

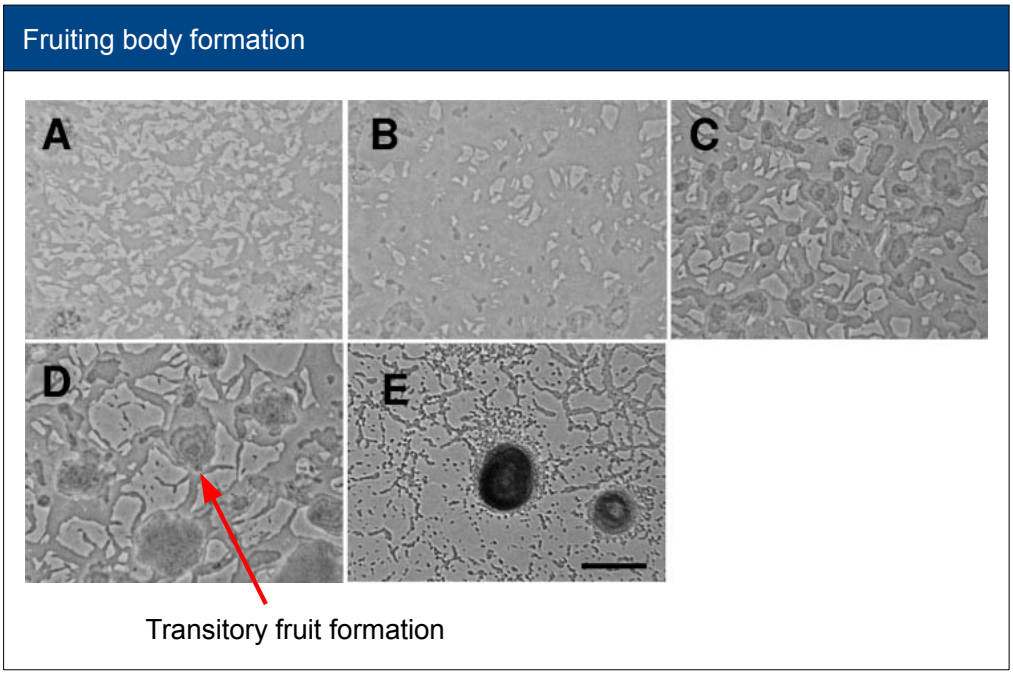


Figure 8.8: Top-down view of fruiting body development showing the aggregation and formation of a fruiting body. Panels C and D show the formation of transitory aggregates. Images adapted from (31).

explained by adjusting the cell influx rate. The simulations maintained the same initial conditions as the previous fruiting simulation, except the rate of influx was altered. A base influx rate was selected to ensure a constantly high cell density to enable fruit formation. Lower influx rates promote transitory fruiting body formation and dissociation. Figure 8.9 shows a snapshot of a simulation where the base influx rate was reduced by 90 %. Although a fruiting body begins to form it rapidly dissociates over time. Cells accumulate and stack outwards from the centre for approximately 200 min after which the fruit collapses and the cells begin to disperse. The cell density remains too low for cells to attempt a new fruit formation suggesting that influx could be a primary driving factor behind development.

Figure 8.10 shows three-dimensional snapshots of fruiting development when the influx rate was reduced to 25 % of the base value. After 500 min, three small mounds have formed; however, they dissociate and new mounds form. This agrees with experimental evidence showing transitory aggregates [31]. If the simulation volume was enlarged by several orders of magnitude (this was not computationally feasible), it is

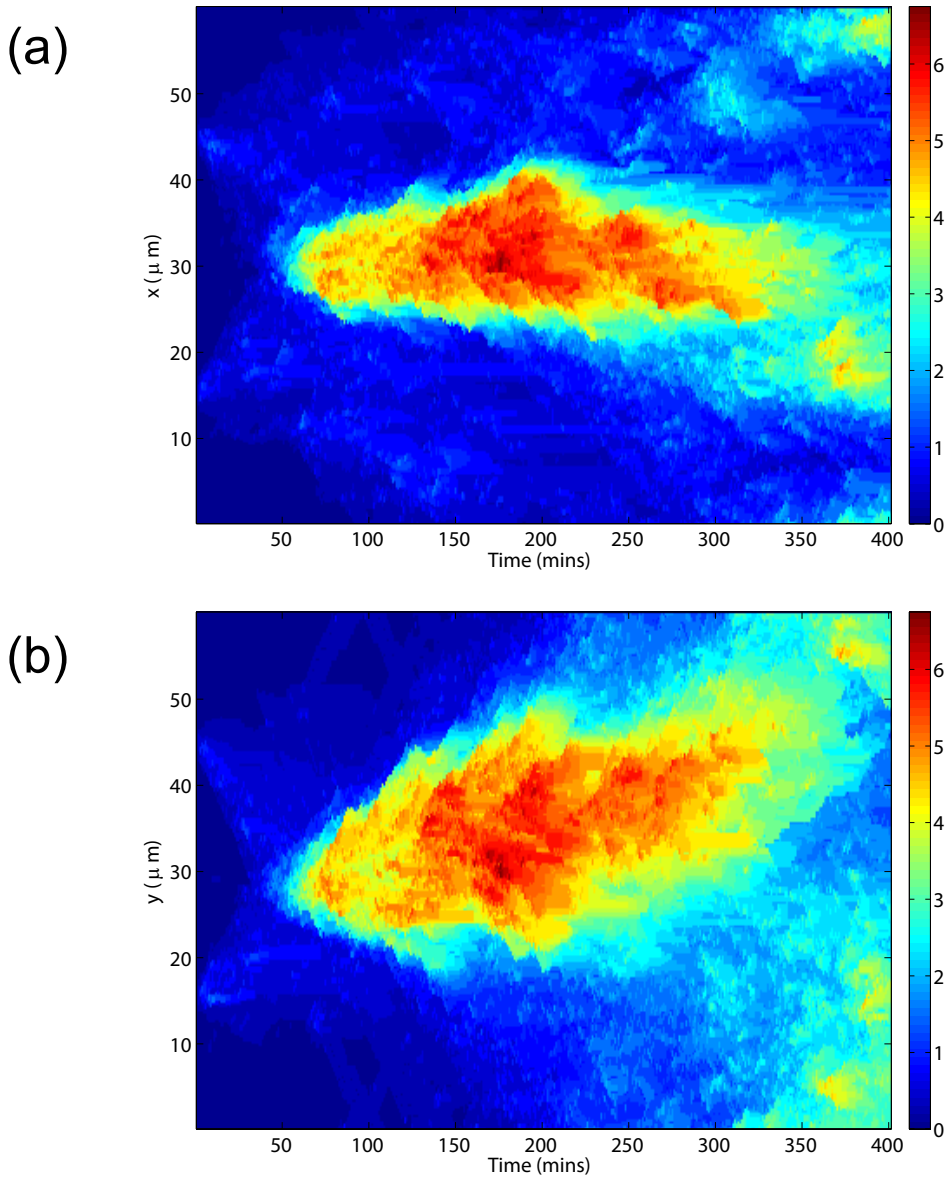


Figure 8.9: Space time plots showing the effect of low cell influx on fruiting body formation. Mound formations are coloured by height from 0 μm (blue) to 6 μm (red). (a) Variation of mound height (z-axis) along the x-axis with respect to time. (b) Variation of mound height (z-axis) along the y-axis with respect to time.

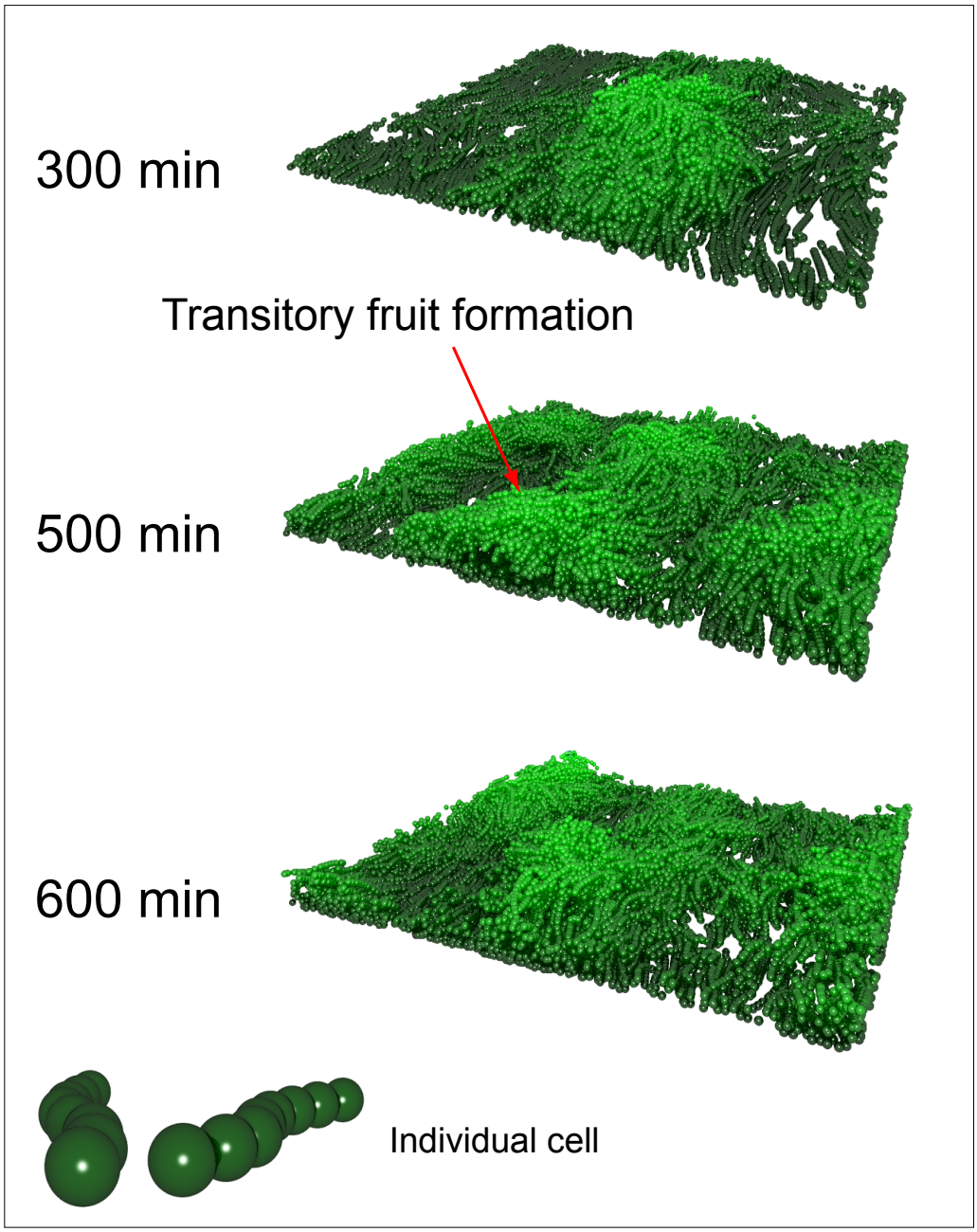


Figure 8.10: Three-dimensional plots showing the how fruiting body formations can spontaneously form and reform. Mound formations are coloured by height from 0 μm (dark green) to 30 μm (light green).

predicted that as fruits disperse, a cohesive layer of cells would form and drift off. This layer would meet other disparate layers from other dispersed fruits and further fruiting development would be initiated where they collide. The process would repeat leading to multiple transitory fruit formations [31]. The prerequisite for this to occur is a sufficiently high cell influx that allows a fruit to form but at a sub-optimal rate such that development cannot be sustained. The fruit must be sufficiently large that when cells leave it, they form a layer of equal density to the initial layers so that fruiting can occur spontaneously at other locations. The influx rate appears to be the rate limiting step in controlling fruiting growth; there is a point where the number of cells forming new layers will begin to exceed the number of cells flowing into the system so the development of new layers is arrested.

8.6.7 Stable fruiting body formation

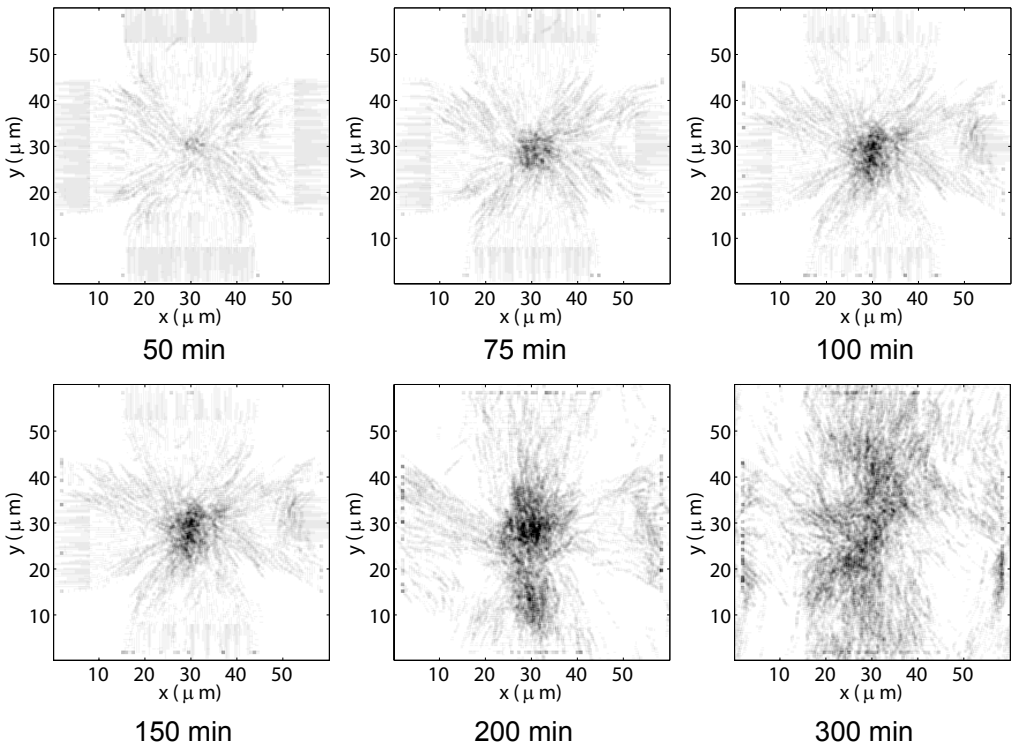


Figure 8.11: Stable fruiting body development in the xy-plane. After 200 min a stable aggregate has formed, which begins to expand outwards.

To assess the ability of the cells to form fruiting bodies, a fruiting model using the

Fruit-Motilator and the influx model described in Section 8.2.3 was constructed. The influx regions were setup so that there was a maximum inflow of cells into the simulation volume; as soon as a swarm had advanced sufficiently, new cells were created to fill all of the resulting space. Figure 8.11 show snapshots of the simulation in the xy-plane. Fruiting development begins after 75 min with a small mound formation. The mound expands outwards as well as upwards forming a large stabilised stalk base after 400 min. The simulation began with 1600 cells initially, but cell influx caused this number to rise to approximately 5000 cells over the duration of the simulation.

Figure 8.12 shows space time plots of how mound formation varies over time in both the x-axis and y-axis. A heat map is used to indicate the maximum height of cells at a given location; blue regions are relatively sparse with cells only a few layers thick whilst red regions contain many cells stacked on top of each other. The highest region is always towards the centre of the simulation volume indicating that cells do the majority of climbing in this region. There is an accumulation of cells, spreading outwards from the centre in both the x-axis and y-axis. The rate of expansion of the fruit from the centre gradually reduces with time. There appears to be a limit on the size of the fruit, a given number of cells can support. The influx rate appears to be the rate limiting step in controlling fruiting growth; there is a point where the number of cells forming new layers will begin to exceed the number of cells flowing into the system so the development of new layers must be arrested.

8.6.8 Sporulation

The motile cell count rises sharply during the first 16 h of simulated real time (see Figure 8.13) with over 5000 cells accumulating within the fruit and surrounding area. After this time point, a combination of the cell density and the aggregate size makes it more difficult for new cells to enter the aggregate. The high cell density ensures constant C-signalling triggering a constant rise in myxospores, which occupy an increasing fraction of the aggregate.

Figure 8.14 shows how mound formation in the fruiting simulation varies over time. Fruiting development begins after 10 h with a small mound formation. This

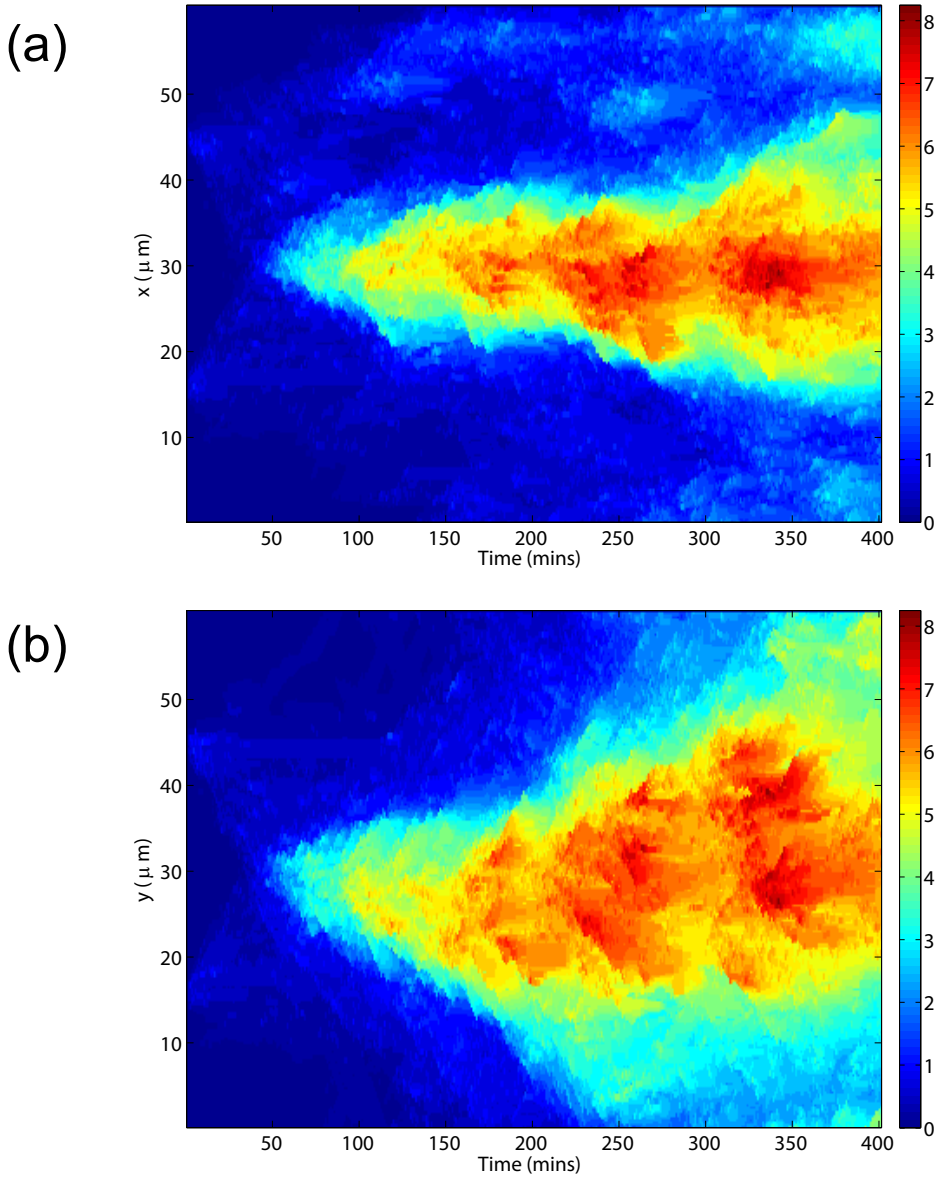


Figure 8.12: Space time plots showing fruiting body formation. Mound formations are coloured by height from 0 μm (blue) to 8 μm (red). (a) Variation of mound height (z-axis) along the x-axis with respect to time. (b) Variation of mound height (z-axis) along the y-axis with respect to time.

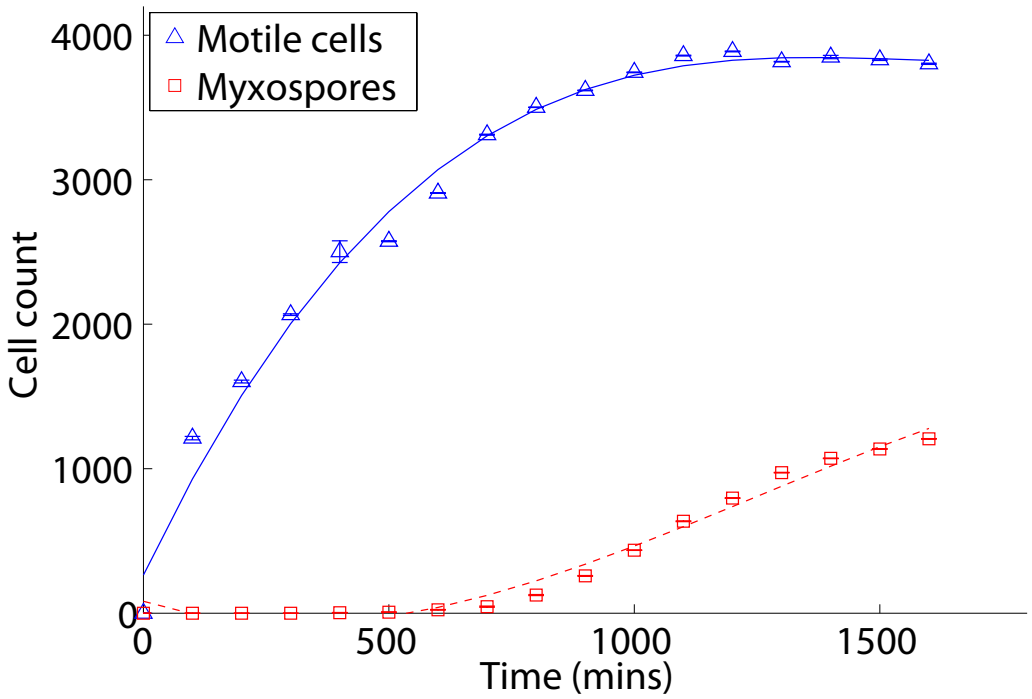


Figure 8.13: Average cell counts of simulated motile and spore cells. The blue triangles and red squares indicate motile myxobacteria cell counts and myxospore counts respectively. Blue solid lines and red dashed lines are the lines of best fit (third degree polynomial) through the motile cell count and myxospore count respectively.

rapidly expands and develops into a fruiting body after 24 h around which motile cells orbit in stream formations. Towards the periphery of the fruit, the cell density rapidly decreases leaving only a thin layer of cells (less than three cells deep) in the regions not occupied by the fruit.

Figure 8.15 shows a fruiting body simulation after 24 h of simulated real time. A large central aggregate has formed in the centre around which cells tend to orbit in stream formations. Cells are highly compact and aligned forming streams and sheets in agreement with the observations of O'Connor and Zusman [118] (see Figure 8.15(a)). The large hemispherical aggregate agrees with the formations observed by Kuner and Kaiser [90] (see Figure 8.15(c)). The aggregate is stable due to the spores that occupy the centre of the mound (see Figure 8.16(a)). Large aggregates of spores inhibit motile cell movement causing them to stall more frequently and slow down and accumulate around the spores driving the aggregate to become bigger. This is in agreement with Sliusarenko et al. [155] who also concluded cell velocity affects fruiting. The mo-

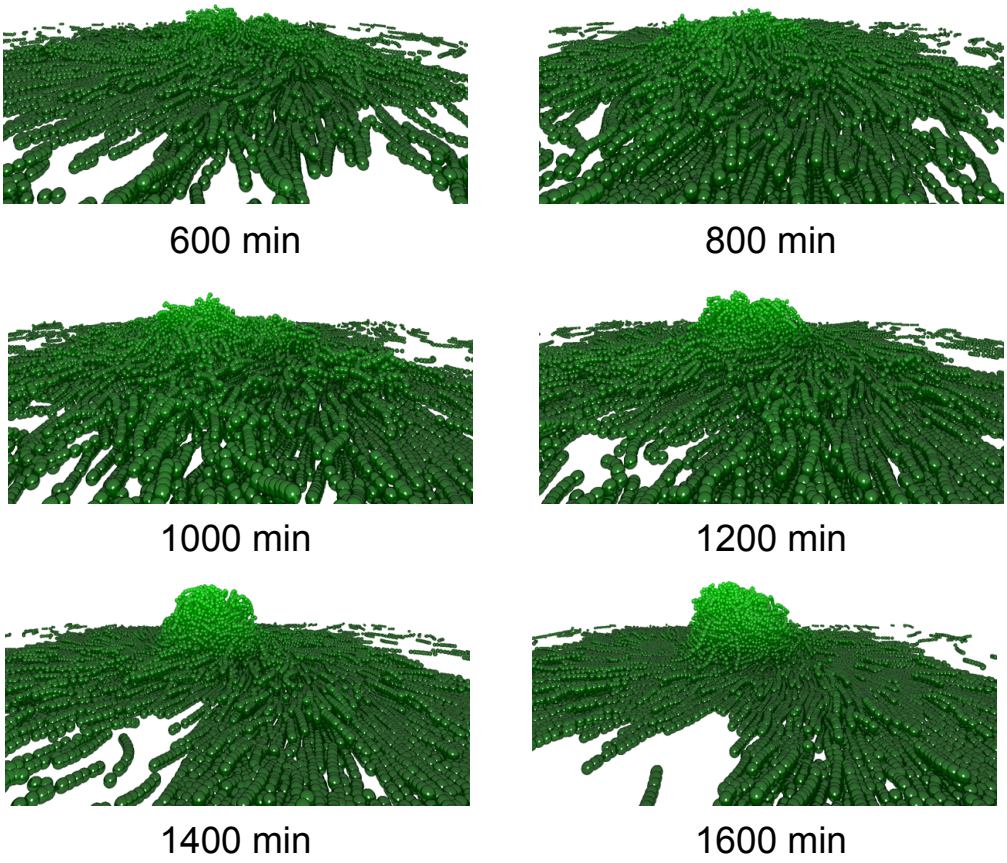


Figure 8.14: Three-dimensional view of fruiting body simulation when cells are allowed to sporulate. After 24 h a single, stable fruit has formed that continues to expand outwards and upwards forming a hemispherical mound.

tile cells push the non-motile cells towards the centre of the aggregate in agreement with existing data [118]. This simulation is available as Movie `mov_fruit_spor` on the `CD-ROM` accompanying the thesis (see Appendix A).

C-signal is not evenly distributed throughout the colony. Cells within the fruit collide much more frequently with other cells so they accumulate C-signal faster (see Figure 8.16(a)) and sporulate faster (see Figure 8.16(b)). The majority of C-signalling occurs within the fruit hence spores are formed within or close to the fruit centre and are pushed into the centre by the motile cells.

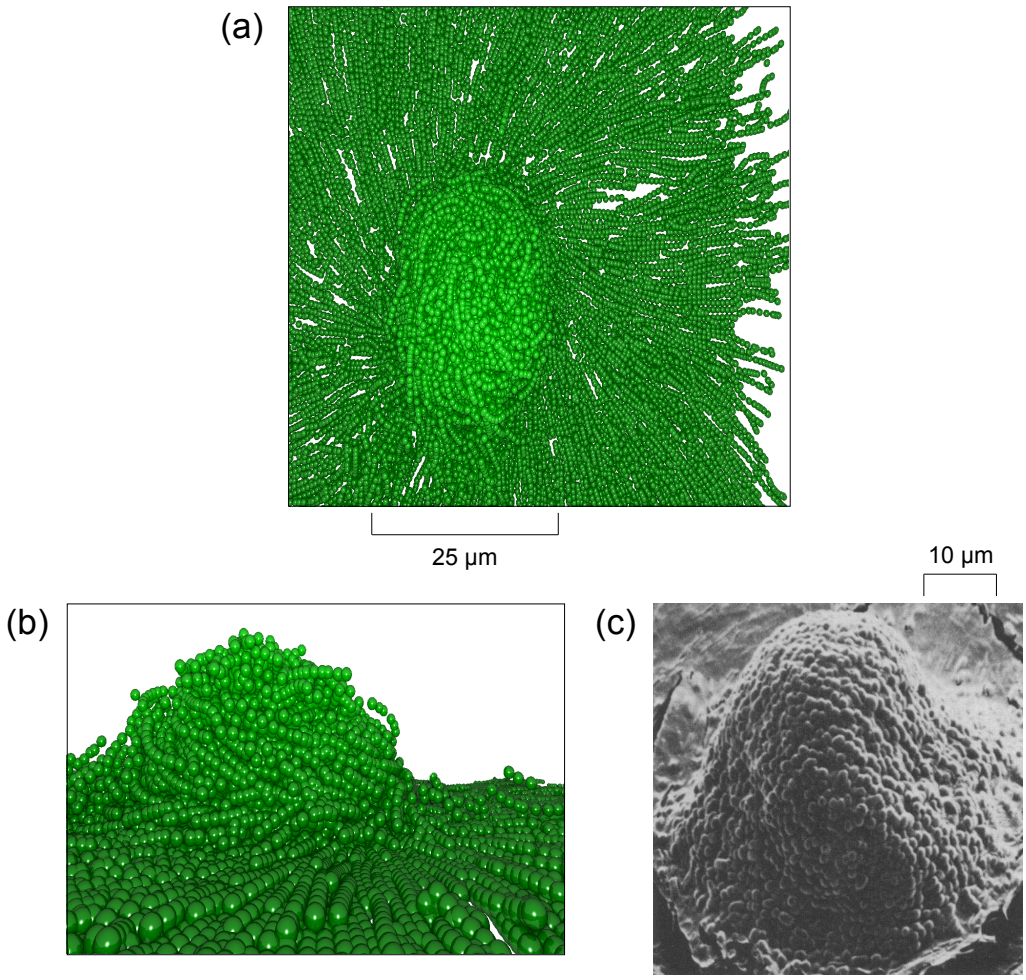


Figure 8.15: View of a fruiting body simulation after 24 h where cells were allowed to sporulate. A stable aggregate has formed around which the motile cells move in streams. (a) Two-dimensional view in the xy-plane. (b) Three-dimensional view of fruiting body. (c) Electron micrograph of a fruiting body adapted from (118).

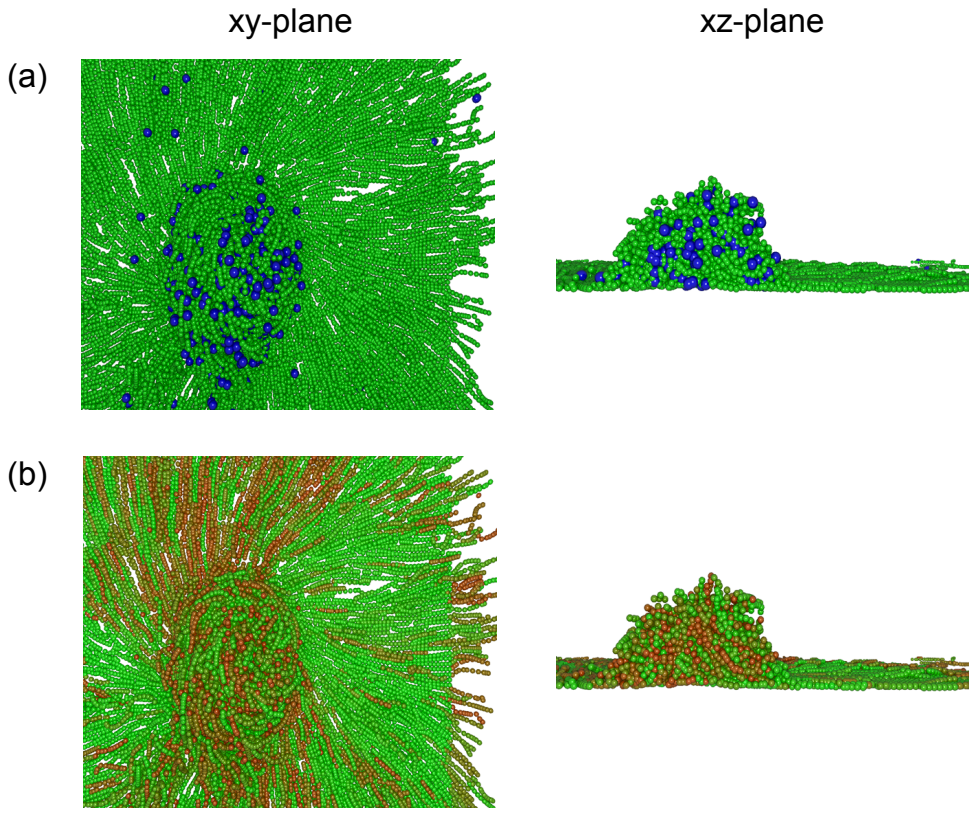


Figure 8.16: The localisation of myxospores and C-signal within a fruiting body. Simulation is shown after 24 h. The fruit corresponds to the formation in Figure 8.14. (a) View of the simulation showing motile cells (green) and myxospores (blue spheres). (b) View of the simulation showing the accumulated C-signal in each cell. C-signal is a dimensionless quantity measured between zero (green) and 400 units (red).

8.7 Discussion

A precise understanding of how and why myxobacteria cells form fruiting bodies remains elusive; however, we can start to address the issue of fruiting body development from a theoretical perspective and provide a possible explanation of how they form. The models presented here indicate that fruit formation can be simply a natural consequence of cell behaviour without any form of centralised coordination. The lack of reversals makes cells prone to collisions [202]. Aggregation centres tend to form quickly as small streams of cells frequently collide and block each other's path. Without the ability to reverse, cells are forced to remain stuck in their current position. Any cells that join the tail of the trail become stuck as well leading to a traffic jam and the

formation of an aggregate. It should be noted that these small streams are typically too small to trigger cells to instigate climbing since although the cells are blocked, the density of neighbouring cells is not generally sufficient to support a new layer of cells on top of it.

Fruiting was addressed by Hendrata and Birner [57] and Sozinova et al. [161]; however their models have a number of limitations. Sozinova et al. use a hexagonal lattice based model, which introduces spatial inaccuracy. O'Connor and Zusman [118] showed that cells cluster in small aligned patches within a fruit and move together. A hexagonal lattice model does not allow for this; cells maintain alignment only if they never change direction, otherwise they alter course by $\pi/3$ radians and spatial alignment is lost almost immediately. Hendrata and Birner use a more spatial realistic model; however, they simulate a relatively small number of cells and must artificially induce aggregation by deliberately stalling a few cells to seed fruit formation. One explanation of why they chose to do this is there are not enough cells in the simulations to form a large enough base to support further layers of cells on top. Even if a few cells do form a new layer, there will never be a high enough density to allow layers to persist; the sparse population dissociates and migrate off the mound. Both models require an artificial aggregate to be formed to seed the fruiting process and both use a climb over model that does not appear to agree with the findings of Curtis et al. [31] showing that the cells form new layers from a centre region that expands outwards [31]

Sozinova et al. [161] used a three-dimensional lattice gas cellular automata model to study rippling formation. Cells were oriented in one of six directions on a hexagonal grid, which introduces spatial inaccuracy. This limits the direction cells can move in and any orbiting patterns of cells may be an artefact of this; any alteration in direction is a turn of $\pi/3$ rad so cells can move through tight arcs. The rigid cell body also means that the cell must alter its course dramatically. In reality, the partial flexibility of the cell means it does not have to completely alter its course to avoid an obstacle; it can bend slightly to align itself alongside the object and move around it. O'Connor and Zusman [118] showed that cells cluster in small aligned patches within a fruit and move toge-

ther. A hexagonal lattice model does not allow for this; cells maintain alignment only if they never change direction, otherwise they alter course by $\pi/3$ radians and spatial alignment is lost almost immediately.

The models presented here show that it is possible for fruiting bodies to develop without artificial induction. Cell density and an upward pushing force seem to be sufficient to instigate formation. Importantly, the EPS surrounding cells must exert an adhesive force, binding cells together. Without this force, cells are too unconstrained and move away from the aggregate. Each layer acts almost independently. Cells from one layer have a much reduced effect on cells in another layer than cells in the same layer. Experiments where all terms in the Hamiltonian were dependent on a local three-dimensional neighbourhood showed that cells cannot move freely due to feeling the effects of cells moving in all directions around them. Consider a scenario where an aggregate has just started to form and a second layer of cells is expanding outwards from the centre (see Figure 8.1, step 2).

Fruiting is the result of a large scale coordination of cells so the effects of scale must be considered in the simulation. In a relatively small cluster of cells (see Figure 8.6), it is relatively easy for cells to move apart, a large proportion of the cells are only one or two cell lengths (cell lengths are used as a measure to negate the differences in scale between biological data and simulated data) away from the colony edge making it easy for them to alter their course. Even small gaps that appear form a large percentage of the total area occupied by the colony (the periphery formed by the outermost cells) leaving a relatively small area upon which new cell layers can form.

The *Fruit-Motilator* is able to explain the initial formation of fruiting bodies from streaming cells as a consequence of cell physics and a low reversal frequency. Fruiting bodies form spontaneously without the need for an artificial aggregation centre to seed the process. The *Fruit-Motilator* has also shown that observed transitory fruiting body developments before a stable fruiting body forms can be explained as a consequence of net cell influx.

The *Spore-Motilator* suggests that sporulation appears to be important for stable fruiting body formation. A large aggregate of non-motile cells provides a base around

which the motile cells can move to form the fruiting body. Motile cells are still driven upwards at the base of the aggregate where streams collide and they force the spores to move upwards as well. This offers a potential mechanism for allowing myxobacterial cells to form sporangiole on stalks without extensive behaviour modifications.

The *Fruit-Motilator* and *Spore-Motilator* offer an explanation of the initial formation of fruiting bodies as a consequence of cell physics and a low reversal frequency, suggesting that fruiting bodies form can form spontaneously without the need for an artificial aggregation centre to start the process. The fruiting model has also shown that observed transitory fruiting body developments before a stable fruiting body forms can be explained as a consequence of net cell influx. Sporulation appears to be important for stable fruiting body formation. A large aggregate of non-motile cells provides a base around which the motile cells can move to form the fruiting body. Motile cells are still driven upwards at the base of the aggregate where streams collide and they force the spores to move upwards as well. This offers a potential mechanism for allowing myxobacterial cells to form sporangiole on stalks without extensive behaviour modifications. Although the model incorporates sporulation, it is still not clear how cells choose to sporulate since only a percentage of the fruiting body do so. The fruiting model approximates this behaviour by only allowing a percentage of the cells to sporulate. Future experimental work will hopefully provide more information on the sporulating process which can be incorporated into the models.

The models presented here offer potential mechanisms *M. xanthus* could use to organise streaming, aggregation and fruiting body formation. Importantly, by deriving the fruiting models from an existing model of rippling, a single model of cell behaviour, based on the observable, physical characteristics of myxobacteria, can explain multiple spatial phenotypes and may shed more light on how myxobacteria is able to exhibit multiple different behaviours during its life-cycle.

Chapter 9

Conclusion

The broad research aim outlined at the beginning of this thesis was to explain morphogenesis and multicellular development in myxobacteria. Specifically, the focus of the work was investigating three important life-cycle stages: rippling, streaming and fruiting. Each stage can be characterised by distinct, complex, spatial pattern formations and myxobacterial colonies cooperatively organising themselves. Together the stages form the core behaviour of myxobacteria in response to starvation and an understanding of them can elucidate how cells can display multiple phenotypes using one set of molecular machinery.

Two approaches were taken to understanding the life-cycle stages: primarily the development of novel mathematical and computational models but also an experimental assay of nutrient use with a view of integrating the data into models. The models help us understand how a relatively simple bacteria can display such apparently complex behaviour. Emphasis was placed on developing a comprehensive model of behaviour that explains multiple behaviours and the transitions between them since current research does not adequately address this issue. Together the research helps to address the following important questions:

- What is the underlying biology governing cell behaviour?
- What is the underlying physics governing cell behaviour?
- How do cells transition from one stage to another?

The computational models necessitated the development of FABCell, a simulation framework to implement and study models with. As well as providing an understanding of the biological behaviour of myxobacteria, an important secondary goal of FABCell was making it a general purpose tool for other researcher to model biological systems. The contributions this thesis has made can be categorised thus:

- An understanding of phosphate acquisition and cell growth.
- The development of novel modelling tools for studying myxobacteria and other systems.
- An understanding of the role of the Frz pathway in rippling.
- An understanding of the physical properties of the cell and how they affect multicellular behaviour.
- An understanding of fruiting body formation as a spontaneous and self-organising process.
- An understanding of the transition between life-cycle stages and how a unified myxobacteria model can display multiple behaviours.

These are explained in more detail.

9.1 Phosphate usage

In Chapter 4, phosphate sources were tested to see if myxobacteria could use them as a nutrient source to promote growth. Pyrophosphate, polyphosphate and glyceraldehyde-3-phosphate were found to reliable growth sources. Interesting myxobacteria cells appear to be fairly discriminating over where they obtain phosphate from. The majority of sources tested, including DNA, seemed unsuitable.

Although this work is distinct from the modelling work that forms the main focus of this thesis, it provides valuable information on nutrition and growth. The models of myxobacteria development are designed to account for important factors governing cell behaviour and it is envisaged that future models will be able to incorporate this data.

9.2 Developing tools to study myxobacteria morphogenesis

The primary focus of this thesis is the development of models to explore development in myxobacteria bacteria. From a theoretical perspective the work was divided into two sections: the development of mathematical models to explain rippling, streaming, and fruiting body formation and the development of tools to implement these models and run simulations on them. There is a *gulf of execution* between how a model is described mathematically and how it is actually implemented in software. Many modelling papers omit or obfuscate technical details (as distinct from the model theory and any results) regarding how the simulations were performed. It is a non-trivial exercise to take a conceptual model and turn it into a working software model that can perform simulations. Although the methods outlined in this thesis are well researched and established techniques for studying physic and biological problems, it proved to be a technical challenge to use them to create models of myxobacteria, especially because no existing software met the requirements of the research, necessitating the development of FABCell. Although initially created to investigate myxobacteria motility, FABCell evolved into a standalone tool in its own right and is now available as open source software [59]. It was decided early in development that one of the deliverables of the project should be a set of tools to help the research community and the realisation of this goal was one of the positive outcomes of the project.

It was envisaged from an early stage that simulations would be large scale and would probably require high performance computing environments to run. This precluded the use of mathematical environments such as MATLAB® or Mathematica, which although excellent tools in their own right, are not suitable for large simulations. They use interpreted script languages for execution, which are slower than natively compiled code for a particular hardware platform and require extra system resources that could otherwise be dedicated to models. MATLAB® in particular has trouble with non matrix operations such as procedural `for` and `while` loops. If procedures must be written in other languages and then compiled as functions suitable for use in these mathematical environments, it defeats the purpose of such tools in the first place.

Excluding the use of mathematical environments left a choice of using a typical scientific programming languages to create FABCell. Java™ has matured steadily over the last ten years and is now a fast, feature rich language that in certain instances can achieve near C level speeds of execution. It was felt that although Java™ is considerably better than it once was, it still requires a large memory overhead to run the Java™ Virtual Machine and code is not running natively. C# from Microsoft has similar problems to Java™ and is constrained to working on the Windows platform.

In the end, C++ was chosen because modern compilers can produce highly efficient code and it is superset (although not strictly) of C that supports all of its features whilst adding an object-oriented paradigm. C++ seemed like a better choice for managing FABCell's large code base. Most of the functionality could be encapsulated within classes themselves within libraries. This meant the system could be apportioned into a number of smaller more manageable sub systems which could be developed almost independently of each other. If changes were necessary in one library it necessitated only the altering and recompilation of one component rather than the whole system, which can be a long and wasteful operation. The majority of the code has remained unchanged so there is no point recompiling it; techniques for achieving such optimisations are desirable.

9.3 A new understanding of the Frz pathway

The *Motilator* model discussed in Chapter 6 offers an understanding of the complexities of the C-signalling pathway and how it is used to control reversals. Crucially it shows that the speculative feedback loop in the *Frzilator* model developed by Igoshin et al. [63] is not required in the signalling pathway. The model establishes a putative mechanism cells might be using to control reversal and also the translocation of FrzS (and RomR) between cell poles. Unlike the *Frzilator*, the *Motilator* allows the effects of nutrients [10] and AglZ [100] on FrzCD to be incorporated into the model (see Section 7.3). This is important in attempting to explain the paradox of how C-signal can induce rippling even though as levels rise within the cell, it actually reduces reversal frequency. Although the core Frz network has been elucidated, the exact mechanism

of how this is translated into reversals has not. The *Motilator* provides evidence of a potential mechanism using a minimal set of assumptions to explain what is happening further downstream of the Frz signalling pathway.

9.4 Modelling cell physics

The *Phys-Motilator* model discussed in Chapter 7 is a complement to the *Motilator* and was designed to address the somewhat artificial constraints imposed by previous lattice models and show that rippling is a truly spontaneous, emergent behaviour. Cell motility was considered from both an intra-cellular (microscopic) and extra-cellular (macroscopic) perspective. The physical properties of a cell are as important as the biological control systems within it; a model of cell motility is useless if the body dynamics are wrong. The *Phys-Motilator* combines the *Motilator* with a physical model of the cell body simulated using Monte Carlo dynamics. By introducing appropriate terms into the system Hamiltonian, the major features of myxobacteria cell motility were captured. The *Phys-Motilator* showed how rippling can manifest through the combined effect of both systems. Rippling is dependent on C-signalling but also the effects of slime trail following, motility and the ability of cells to flex and climb. The combination of these factors are crucial for multicellular development; without them, experiments showed that cells are incapable of spatial pattern formations.

9.4.1 Model realism

FABCell could be scaled into a supercomputing environment to facilitate much larger simulations. There is a perpetual problem in computational modelling; hardware rarely matches the complexity of the simulation so certain sacrifices must be made. The models presented in this thesis were balanced between computational feasibility and the real world realism they offered. The effect of Moore's Law[1] means that processing power continually increases[2] so ever more complex models can be simulated. Limi-

[1] A trend observed by Gordon Moore (co-founder of Intel®) showing that the number of transistors that can be placed on an integrated circuit doubles approximately every two years.

[2] This is not strictly true as the physical limit of processing speed that can be obtained on silicon based processors is being rapidly being reached.

tations of the models presented here will not be an issue within a few generations of hardware development.

Hendrata and Birnir [57] simulated on average 300 cells in their simulation of fruiting. Wu et al. [200] appear to simulate similar numbers of cells although they do not state explicitly the size of the simulation. Their three node cell model is also open to question because having three nodes restricts movement and flexibility. The models presented here involved over 10,000 cells using Monte Carlo methods whilst the segmented model used in Chapter 7 and Chapter 8 offers a higher degree of freedom similar to Starruß et al. [165] but with a computation speed comparable to Wu and Hendrata.

9.5 Fruiting

The logical extension of the *Phys-Motilator* is the *Fruit-Motilator* described in Chapter 8. Like the *Motilator* and *Phys-Motilator*, its development was partially motivated by a desire to address the limitations of the few existing models of fruiting development.

The model shows that the layer building concept proposed by Curtis et al. [31] is a sufficient and plausible mechanism for fruiting development. It captures all of the features from this research: layer formation, transient fruiting body development in the initial stages, static fruit develop during sporulation and the spontaneous organisation of spores into the centre of a fruiting body formation. In particular the models shows that fruits do not have be started from an artificial aggregation centre as in Hendrata and Birnir [57].

The *Fruit-Motilator*also questions the previously unchallenged model [57, 161] that assumes fruits form by cells climbing over a static aggregation centre. This model is based on research from two papers published over 20 years ago [90, 118] which are inconclusive. In the interim period it has not been investigated further. The research from Curtis et al. offers much more compelling biological evidence that fruit formation starts with mono-layers of cells colliding and pushing up without aggregation centres which remain static for the duration of the fruit. There is little evidence to support the idea of cells stalling causing other cells to begin climbing. Cells move continuously

and have never been shown to remain stationary for extended durations. Cells are clearly capable of climbing, otherwise the fruit would not form. It follows that cells in the middle of a collision would attempt to climb rather than arbitrarily stalling for no apparent reason.

Modelling fruiting development is far from complete. The *Fruit-Motilator* does not currently explain the intricate fruiting formations in some of the myxobacteria species. *M. xanthus* has a relatively simple fruiting body formation compared to other species such as *Stigmatella aurantiaca* and *Chondromyces crocatus* (see Figure 2.6) [134] where the fruits are more complicated and feature much more prominent stalk formations. Cells effectively differentiate into stalk and sporangeolie cells. The model does not account for this and would require more experimental data to ascertain how cells control their position and function within the fruit. The models are already capable of clustering and moving spores with the fruit so the key is understanding from a biological and theoretical perspective, how spores are kept together to form the head of the fruit and how cells can form very tall, thin stalk formations without cells collapsing.

Sporulation is still not fully understood. The *Spore-Motilator* shows that cells, which are triggered to sporulate by C-signal, will cause fruits to develop but it does not address the mechanisms used to limit the number of cells that will sporulate. Using the *Spore-Motilator*, a steady influx of cells into the fruit will cause the C-signal levels to rise indefinitely leading to a fairly constant sporulation rate. In reality only 10 % of cells ever sporulate so a form of quorum sensing or nutrient depletion may be limiting the sporulation rate. Further work will be required to determine an accurate measure of sporulation rates and also how cells coordinate sporulation so that only a minority ever do so.

9.6 A comprehensive approach

The models presented here show that a well characterised model of myxobacteria cell motility can describe rippling, streaming and fruiting and cells do not require multiple cellular control systems to display different phenotypes. This validates both the model itself and what is known biologically about myxobacteria. Neither rippling nor fruiting

can be explained purely by the function of signalling pathways alone; it is also a direct result of the physical characteristics of the cell. My work serves as a basis for future models looking at other aspects of the myxobacterial life-cycle and more generally as a model of complex emergent behaviour.

By integrating a simple model of C-signalling with a detailed Hamiltonian capturing the physics of single cell motility, it was possible to explain how a cell population could self-organise to display multiple different spatial phenotypes. Importantly the work presented in the thesis has shown that a single unified model is sufficient to explain multiple behaviours where previously multiple different models were required to explain each state of the life-cycle. Together, Chapter 6, Chapter 7 and Chapter 8 form a cohesive model of myxobacteria multicellular development that can explain each of the major life-cycle events and the transition from each phase to the next, which previous models have failed to do.

The myxobacteria are complex organisms and models can never completely describe a system fully. By their very nature, models are approximations of a real system that can provide compelling evidence of how it might function. Models are important for validating concepts and ideas and showing that they work in theory based on all the evidence available. This thesis is mostly concerned with the use of models to explain phenomena and therefore a critique of models in general is necessary for balance and to admit that there are limitations and caveats, as with any process. Nevertheless, it is hoped that the models presented in this thesis shed more light on myxobacterial behaviour.

9.7 Future Work

9.7.1 A biological perspective

A primary focus of current myxobacteria research is understanding the Frz signalling pathway since it is not really well understood [205]. It was recently shown that FrzCD appears to relocate within a cell to direct the motility machinery [99]. Maureillo et al. [100] recently showed that AglZ appears to inhibit FrzCD activation. The *Motilator*

supports these findings.

The cytoskeletal track proposed by Mignot et al. [110] will need further investigation to verify the *Motilator* as well as experiments to determine if there are biological switches controlling FrzS and RomR recruitment and how they function.

The focal adhesion mechanism proposed by Mignot is an alternative to the slime propulsion mechanism that is thought to control motility [108]. There is evidence to suggest cells have helical extracellular adhesion tracks which form a groove in the slime. The cell rotates and moves using screw propulsion. This mechanism has not conclusively been shown to be the propulsion mechanism cells use; however, it would be interesting to model cell motion using this idea and compare it with the slime extrusion models.

Although the phosphate assay was largely separate to the modelling component of thesis, the data is yielded answers questions about nutrient use and development. This data will hopefully be incorporated or guide future models looking at regulated development.

9.7.2 A computational perspective

FABCell will hopefully prove to be a useful tool to the biological and modelling research communities. As an academic piece of software, certain aspects of the framework were given more emphasis that others. Notably the user interface does not currently match desired ideals. It was initially planned that FABCell would have a windowed user interface that users could use to implement models. This would have been particularly useful for non-technical users who do not wish to interact directly with FABCell or learn how to program and develop with it. It was not possible to create such an interface in this manner although a great number of strides were made towards improving usability.

The plug-in extension mechanism (see Section 5.8) allows the system to be extended to cope with new functionality without requiring any major system changes. This is beneficial to more experienced users but could have a negative impact on normal users. A graphical user interface that allows users to point and click to start simu-

lations has limitations. Whilst a lot of the inherent complexity can be masked from the user so it is easy to use, it simultaneously limits what they can use the system for. Having a unified interface to support all of the potential plug-ins the system grow to include could ultimately prove difficult. A future project to develop such a user interface would undoubtedly improve usability. FABCell was started at the lowest and most complex level[3] with each further revision adding layers of abstraction for particular functionality or ease of use. This bottom up approach best suited the nature of the project since new functionality as necessary without going through a long cascade of starting with a complex user interface and working back redesigning each layer to add the new functionality.

A completely non-programming paradigm is not realistically achievable except in a few special cases. To shield users from a programming language such as C++ typically requires that user input be parsed in some fashion and then translated into something more suitable for the FABCell architecture. Parsing and rewriting instructions for a lower level system is essentially the role of an interpreted language such as can be found in MATLAB®, Mathematica® and PERL and as stated earlier, this was avoided precisely because of the computational overhead it adds. However, certain cellular automata are suitable for transition to no programming especially those that use two state logic where the outcome is binary based on a set of binary inputs. The Game of Life model discussed in Section 5.11.1 does not require any programming on the user's part; only the setup parameters need to be specified. Allowing FABCell to be controlled using an XML schema partially addresses the *gulf of execution* between using FABCell and requiring an extensive knowledge of the API. It is intuitive and easy to use method to create simulations. Future revisions of FABCell will make the user interface more user friendly and allow more options to be configured through a windowed interface making it even easier to use.

[3]Levels in software development terms refer to the level of abstraction the system presents the user. The greater the abstraction, the more hiding of details and generally the more user-friendly a system is. Programming in C++ would be considered relatively low level given the esoteric knowledge required to use it whereas a graphical user interface, such as for programs in Microsoft® Windows®, would be considered high-level because it shields users for the underlying complex sub-systems.

Appendix A

Supplementary movies

A selection of movies of myxobacteria cell developmental events *in vivo* and the models presented in this thesis are available on the accompanying CD-ROM. Table A.1 and Table A.2 list the location of the movies on the CD-ROM.

Table A.1: Myxobacteria movies

Movie:	mov_chon_fruiting
Name:	*Chondromyces* fruiting
File:	chondromyces_fruiting.mov
Directory:	/movies/bio/
Time lapse video of *Chondromyces* forming fruiting bodies.	
Movie:	mov_development
Name:	*M. xanthus* development
File:	development.mov
Directory:	/movies/bio/
M. xanthus development showing cells forming fruiting bodies in the *xy*-plane.	
Movie:	mov_germination
Name:	Germination
File:	germination.mov
Directory:	/movies/bio/
Spores germinate into vegetative cells.	
Movie:	mov_ripples
Name:	*M. xanthus* rippling
File:	ripples.mov
Directory:	/movies/bio/
M. xanthus cells rippling in the *xy*-plane.	
Movie:	mov_slime_trails
Name:	Slime trails
File:	slime_trails.mov
Directory:	/movies/bio/

Myxobacteria cells deposit slime trails and preferentially follow the slime trails left by other cells.

Movie:	`mov_swarming`
Name:	Swarming
File:	`swarming.mov`
Directory:	`/movies/bio/`

Myxobacteria cells swarming.

Table A.2: FABCell movies

Movie:	`mov_rule_110`
Name:	Wolfram Rule 110 CA
File:	`rule_110.zip`
Directory:	`/movies/fabcell/ca/rule_110/`

An implementation of the Wolfram Rule 110 CA [194].

Movie:	`mov_gol_gun`
Name:	Game of Life Gun
File:	`gol_gun.zip`
Directory:	`/movies/fabcell/gol/gun/`

An CA implementation of the Game of Life demonstrating a *gun* (allows the population to grow by repeatedly shooting out moving objects).

Movie:	`mov_gol_qbs`
Name:	Game of Life Queen Bee Shuttle
File:	`gol_qos.zip`
Directory:	`/movies/fabcell/gol/qbs/`

A CA implementation of the Game Of Life demonstrating the *queen bee shuttle*.

Movie:	`mov_cell_sort`
Name:	CPM cell sorting
File:	`cell_sort.zip`
Directory:	`/movies/fabcell/cell_sort/`

A CPM implementation of the cell sorting model described by Graner and Glazier [49].

Movie:	`mov_mot`
Name:	*Motilator*
File:	`motilator_ca.mov`
Directory:	`/movies/fabcell/motilator/`

An LGCA model of myxobacteria cell rippling using the *Motilator* as described in Chapter 6.

Movie:	`mov_mot_no_coll`
Name:	*Motilator* (no collisions)
File:	`motilator_no_coll.mp4`
Directory:	`/movies/fabcell/motilator/`

An LGCA model of myxobacteria cell rippling without collisions using the *Motilator* as described in Chapter 6.

Movie:	`mov_phys_mot`
Name:	*Phys-Motilator*
File:	`phys_motilator.mov`

Directory: `/movies/fabcell/phys_motilator/`

A combined model of myxobacteria cell rippling using the *Motilator* and a Monte Carlo model of cell dynamics as described in Chapter 7.

Movie: `mov_fruit`
Name: *Fruit-Motilator*
File: `fruit_motilator.mp4`
Directory: `/movies/fabcell/fruit/`

A Monte Carlo model of fruiting body development as described in Chapter 8.

Movie: `mov_climbing`
Name: Climbing cells
File: `climbing.mp4`
Directory: `/movies/fabcell/fruit/`

A simulation of the fruiting cells described in Chapter 8 showing how they climb over obstacles (some static cells).

Movie: `mov_fruit_tran`
Name: *Fruit-Motilator*(transitory fruits)
File: `fruit_motilator_tran.mp4`
Directory: `/movies/fabcell/fruit/`

A Monte Carlo model of fruiting body development with sporulation as described in Chapter 8.

Movie: `mov_fruit_spor`
Name: *Spore-Motilator*
File: `fruit_motilator_spore.mp4`
Directory: `/movies/fabcell/fruit/`

A Monte Carlo model of fruiting body development with sporulation as described in Chapter 8.

Appendix B

Motilator equations

B.1 General form of the Jacobian matrix for the *Motilator*

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial m} & \frac{\partial f_1}{\partial s_1} & \frac{\partial f_1}{\partial s_2} & \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial m} & \frac{\partial f_2}{\partial s_1} & \frac{\partial f_2}{\partial s_2} & \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \\ \frac{\partial f_3}{\partial m} & \frac{\partial f_3}{\partial s_1} & \frac{\partial f_3}{\partial s_2} & \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} \\ \frac{\partial f_4}{\partial m} & \frac{\partial f_4}{\partial s_1} & \frac{\partial f_4}{\partial s_2} & \frac{\partial f_4}{\partial x_1} & \frac{\partial f_4}{\partial x_2} \\ \frac{\partial f_5}{\partial m} & \frac{\partial f_5}{\partial s_1} & \frac{\partial f_5}{\partial s_2} & \frac{\partial f_5}{\partial x_1} & \frac{\partial f_5}{\partial x_2} \end{pmatrix} \tag{B.1}$$

B.2 Jacobian matrix for the *Motilator*

The partial derivatives of Equation B.1 are as follows. Terms not specified have a derivative of zero.

$$\frac{\partial f_1}{\partial m} = \frac{zk_m}{(K_m + (1-m))^2} - \frac{mzk_m}{(K_m + (1-m))^2} - \frac{zk_m}{K_m + (1-m)} + \tag{B.2}$$

$$\frac{mk_{d_m}}{(K_{d_m} + m)^2} - \frac{k_{d_m}}{K_{d_m} + m} \tag{B.3}$$

$$\frac{\partial f_2}{\partial m} = \frac{k_1 x_1 (1-s_1)}{K_1 + (1-s_1)} - \frac{k_{d_1} x_2 s_1}{K_{d_1} + s_1} \tag{B.4}$$

$$\frac{\partial f_2}{\partial s_1} = \frac{k_1 x_1 m}{(K_1 + (1-s_1))^2} - \frac{k_1 x_1 m s_1}{(K_1 + (1-s_1))^2} - \frac{k_1 x_1 m}{K_1 + (1-s_1)} + \tag{B.5}$$

$$\frac{k_{d_1} x_2 m s_1}{(K_{d_1} + s_1)^2} - \frac{k_{d_1} x_2 m}{K_{d_1} + s_1} \tag{B.6}$$

$$\frac{\partial f_2}{\partial x_1} = \frac{k_1 m (1-s_1)}{K_1 + (1-s_1)} \tag{B.7}$$

$$\frac{\partial f_2}{\partial x_2} = \frac{-k_1 m s_1}{K_{d_1} + s_1} \tag{B.8}$$

$$\frac{\partial f_3}{\partial m} = \frac{k_2 x_2 (1-s_2)}{K_2 + (1-s_2)} - \frac{k_{d_2} x_1 s_2}{K_{d_2} + s_2} \tag{B.9}$$

$$\frac{\partial f_3}{\partial s_2} = \frac{k_2 x_2 m}{(K_2 + one - s_2)^2} - \frac{k_2 x_2 m s_2}{(K_2 + one - s_2)^2} - \frac{k_2 x_2 m}{K_2 + (1-s_2)} + \tag{B.10}$$

$$\frac{k_{d_2} x_1 m s_2}{(K_{d_2} + s_2)^2} - \frac{k_{d_2} x_1 m}{K_{d_2} + s_2} \tag{B.11}$$

$$\frac{\partial f_3}{\partial x_1} = \frac{-k_2 m s_2}{K_{d_2} + s_2} \tag{B.12}$$

$$\frac{\partial f_3}{\partial x_2} = \frac{k_2 m (1-s_2)}{K_2 + (1-s_2)} \tag{B.13}$$

$$\frac{\partial f_4}{\partial s_1} = -k_1^{max} x_1 \tag{B.14}$$

$$\frac{\partial f_4}{\partial x_1} = -d_1 - s_1 k_1^{max} \tag{B.15}$$

$$\frac{\partial f_4}{\partial x_2} = \frac{-\alpha_1 x_2^{p-1} p}{(\beta_1 + x_2^p)^2} \tag{B.16}$$

$$\frac{\partial f_5}{\partial s_2} = -k_2^{max} x_2 \tag{B.17}$$

$$\frac{\partial f_5}{\partial x_1} = \frac{-\alpha_2 x_1^{q-1} q}{(\beta_2 + x_1^q)^2} \tag{B.18}$$

$$\frac{\partial f_5}{\partial x_2} = -d_2 - s_2 k_2^{max} \tag{B.19}$$

Appendix C

MATLAB® implementation of the *Frzilator*

C.1 ODE system

MATLAB®'s `ode45` function is used to perform the integration step.

```
global t0;
global k2a;
global deltaT;

k2a = 0.5;
t0 = 10;
deltaT=0.5;

tspan=[0 100];
x0 = [0 0 0];

[t,x]=ode45(@oster_orig, tspan, x0);

hold off;
plot(t(:,1),x(:,1), '-k', t(:,1), x(:,2), '--k', t(:,1), x(:,3), ':k')
set(gca, 'FontSize', 24);
legend('frzf', 'frzcd', 'frze', 'location', 'NorthWest');
xlabel('Time (mins)');
ylabel('Concentration');
```

C.2 ODE step

```
function xprime = oster_orig(t, x)
```

```
% x(1) is f, x(2) is c, x(3) is e

global t0;
global k2a;
global deltaT;

Ka = 1e-2;
Kd = 5e-3;
Km = 5e-3;
Kdm = 5e-3;

Kp = 5e-3;
Kdp = 5e-3;

k1a = 0.08;%0.9;

kdmax = 1;

kmmax = 4; %4;
kdmmax = 2; %2;
kpmax = 4; %4;
kdpmax = 2; %2;

kamax = k1a + (k2a * (heaviside(t - t0) - heaviside(t - t0 - deltaT)));

ka = kamax / (Ka + (1 - x(1)));
kd = kdmax / (Kd + x(1));

km = kmmax / (Km + (1 - x(2)));
kdm = kdmmax / (Kdm + x(2));
kp = kpmax / (Kp + (1 - x(3)));
kdp = kdpmax / (Kdp + x(3));

xprime = [(ka * (1 - x(1))) - (kd * x(1) * x(3));
    (km * (1 - x(2)) * x(1)) - (kdm * x(2));
    (kp * (1 - x(3)) * x(2)) - (kdp * x(3))];
```

C.3 System response

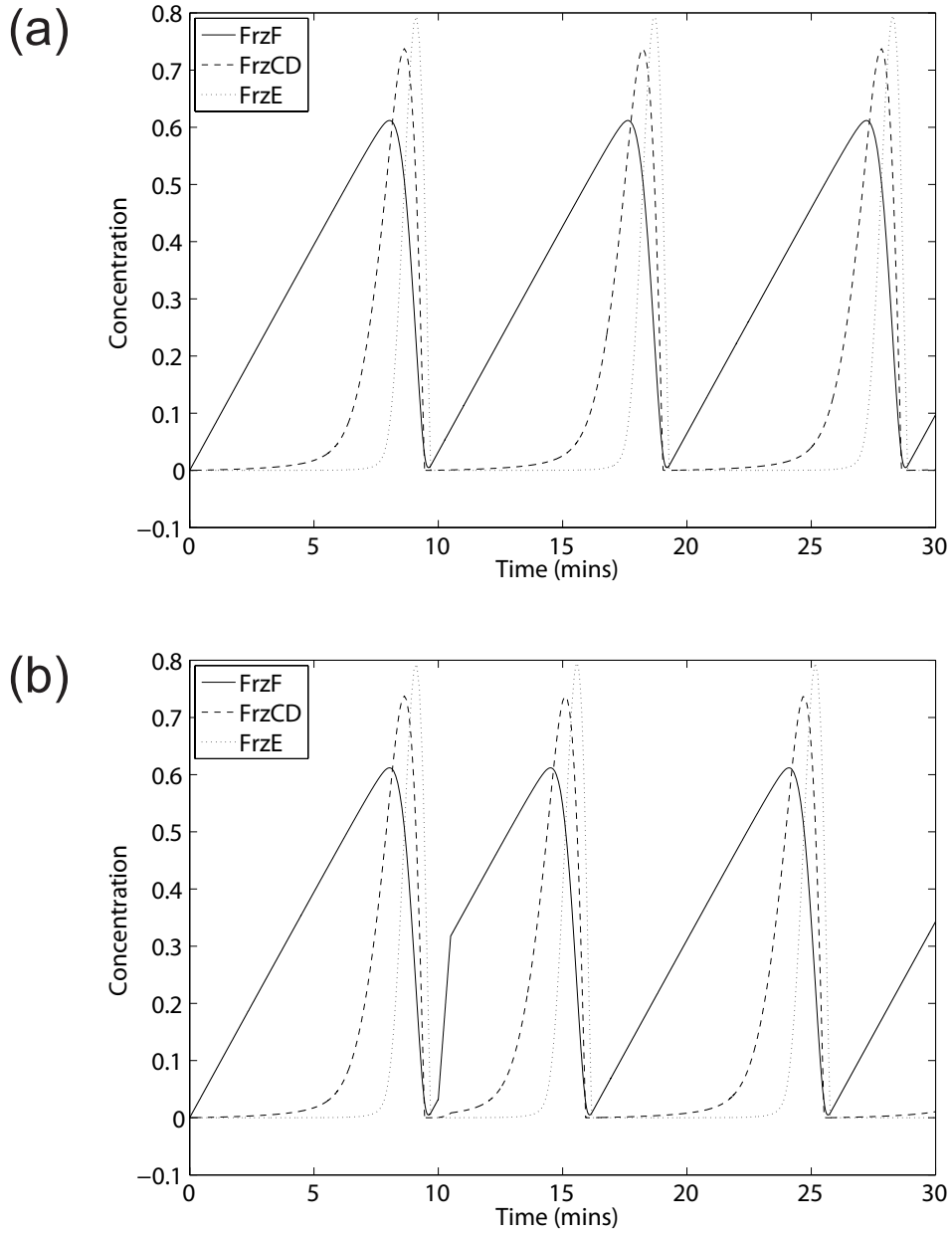


Figure C.1: Oscillations of the *Frzilator*. (a) Unperturbed *Frzilator* (see Figure 2(a) in (63)). (b) A 0.5 min pulse of signalling at $t = 10$ min causes a speed up in reversal frequency (see Figure 2(b) in (63)).

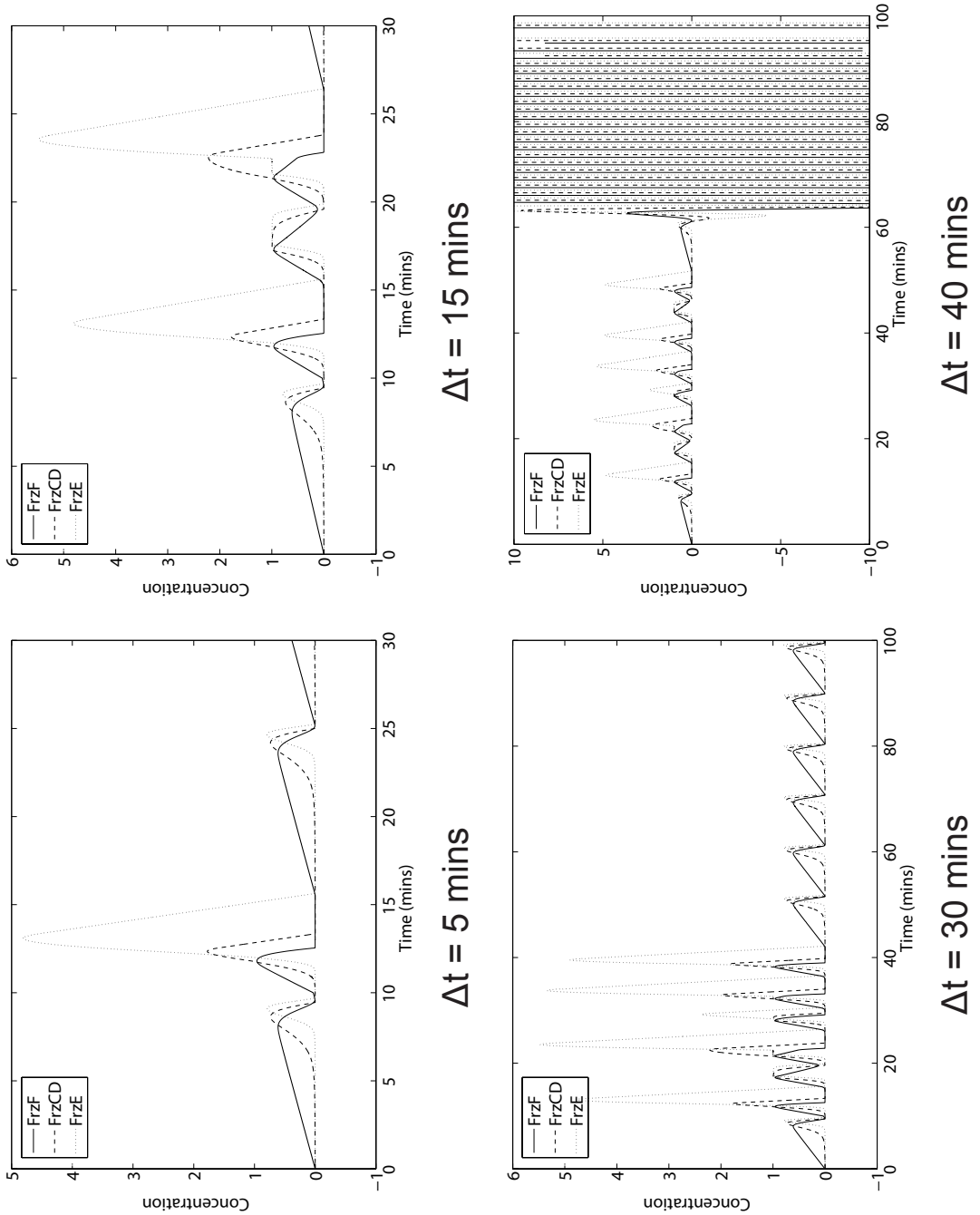
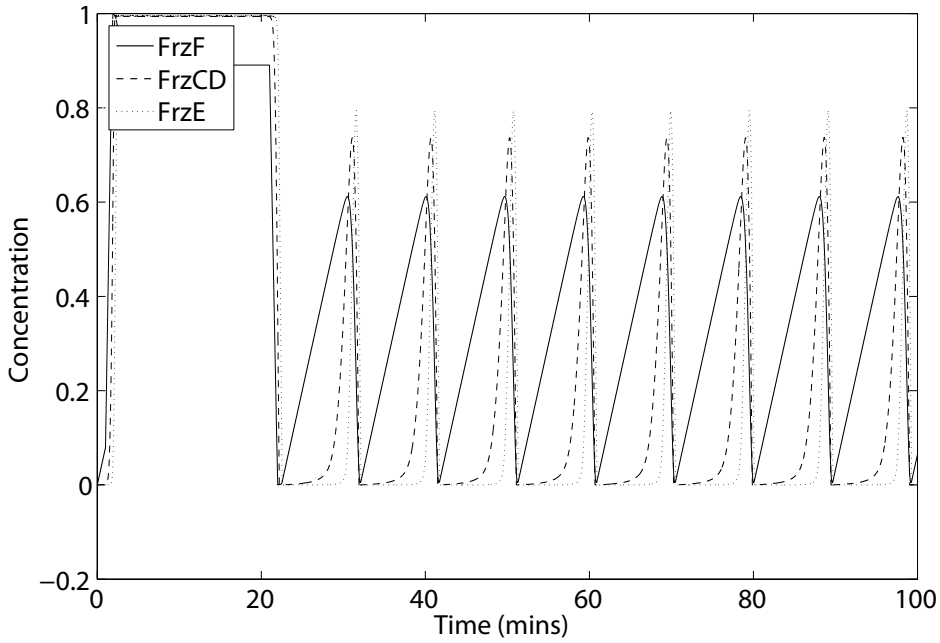
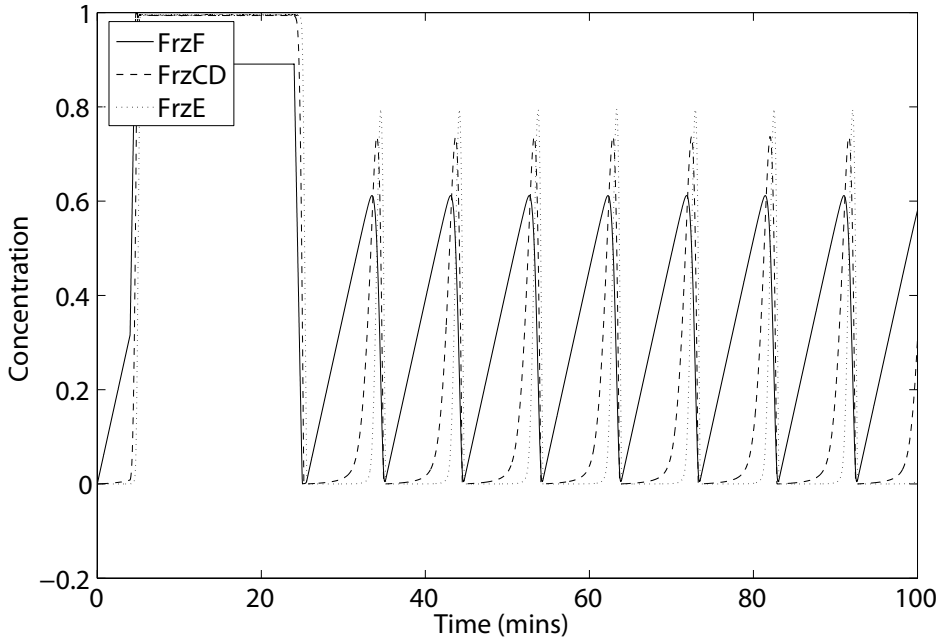


Figure C.2: *Frzilator* response to increases in C-signalling duration. Using the parameters specified in Table 1 in (63), bursts of C-signal with increasing duration were given to the system. The response is inconsistent and when $\delta t = 40$ min, the system becomes chaotic.

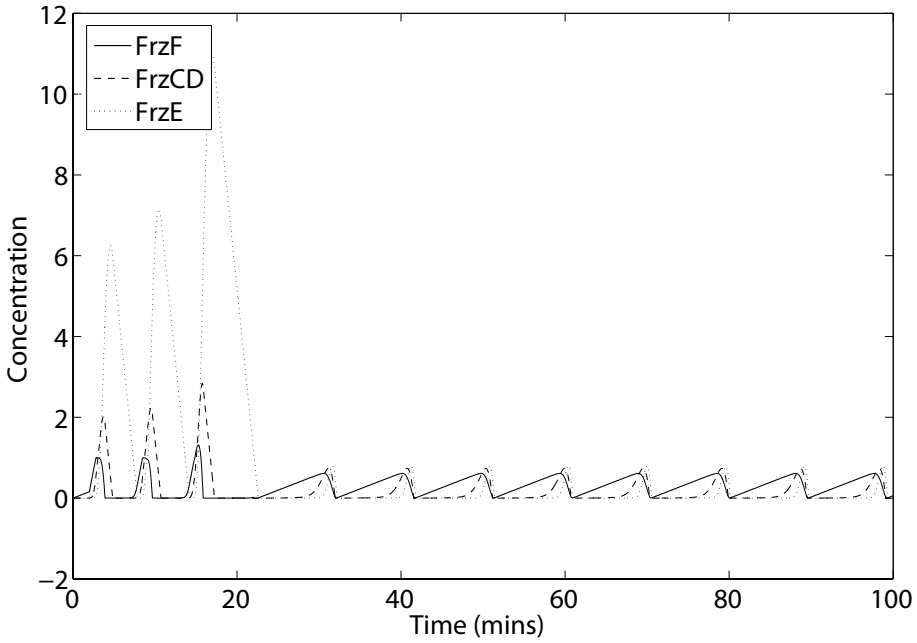


t_0 = 1 min

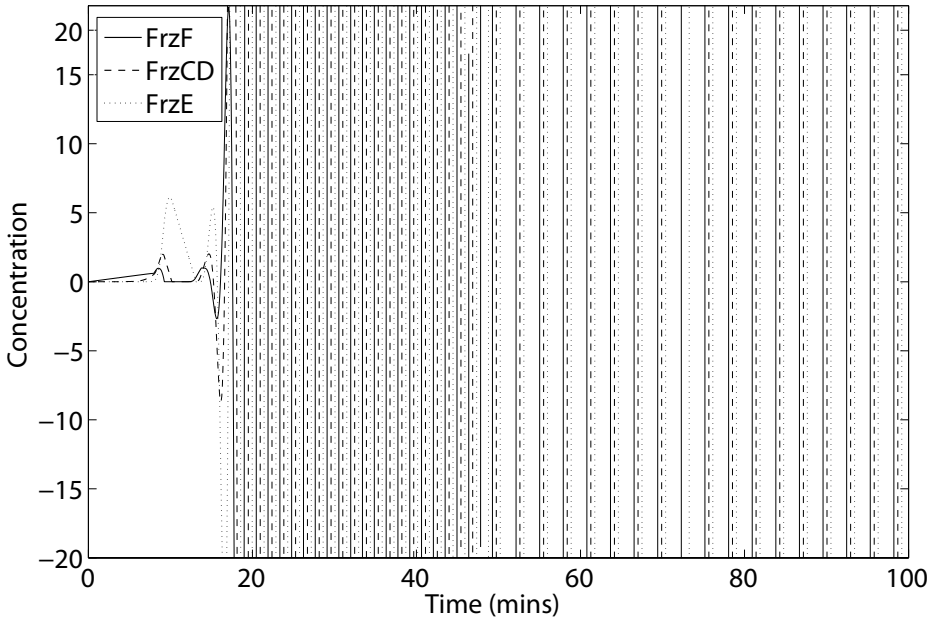


t_0 = 4 min

Figure C.3: Stable *Frzilator* response to sliding 20 min signalling window during the streaming phase. If the signalling begins at $t_0 = 1$ min then the reversal frequency decreases.



t_0 = 2 min



t_0 = 8 min

Figure C.4: Unstable *Frzilator* response to sliding 20 min signalling window during the streaming phase. If the signalling begins at $t_0 = 1$ min then the reversal frequency decreases. When $t_0 = 8$ min, the system becomes chaotic suggesting that signalling events spanning the refractory period cause the system to become unstable.

Appendix D

MATLAB® implementation of the *Motilator*

D.1 ODE system

MATLAB®'s `ode45` function is used to perform the integration step.

```
global pulse0;
global pulse1;
global pulseh;
global pulsel;
global mgla_basal;

x_start = 5;
x_end = 40;

pulse0 = 0;
pulse1 = 0;

tstart = 0;
tend = 100;

tspan=[tstart tend];

x0 = [2 0 one 1 0];

% pulse = 0.15

% pulse background mgla
% max out at 1
```

```
pulseh = 0.75;
pulsel = 0.08;
mgla_basal = pulsel;

[t, x] = ode45(@reverse_mechanism_10, tspan, x0);
```

D.2 ODE step

```
function xprime = reverse_mechanism_10(t, x)

global pulsel;
global pulseh;
global pulse0;
global pulse1;
global mgla_basal;



a1 = 3;
a2 = 3;
b1 = 1;
b2 = 1;

% need to keep it switched off otherwise it reverts



%adjust initial response to stimulus
%lower p => faster response
p1 = 3;
p2 = 3;

Ks1 = 1e-3;
Ks2 = Ks1;

%set these lower than a1max etc
% controls min oscillation time bound
ks1max = 0.8;%0.8;
ks2max = ks1max;

done = 1;
d2 = done;
```

```
s1 = done + (x(3) * ks1max);
s2 = d2 + (x(4) * ks2max);

frzs_switch_2_1 = x(1);
frzs_switch_1_2 = x(2);

% adjust slope of decent of frzs
Kh = 0.01;
Kt = Kh;

%mgla
m2 = m1;

% need to adjust x(5) so the base line is approx 0.15
m_in_1 = x(5) * 0.5;
m_out_1 = x(5) * 0.5;

m_in_2 = m_in_1;
m_out_2 = m_out_1;

%mgla is needed for proper localisation
Kh = 0.01;
Kt = Kh;

frzsk = 1;
frzsK = 0.05;
frzsKd = frzsK;

frzskd = 1;

frzs_in_1 = m1 * (1 - x(3)) * frzsk / (frzsK + (1 - x(3)));
frzs_out_1 = m2 * x(3) * frzskd / (frzsKd + x(3));

frzs_in_2 = m2 * (1 - x(4)) * frzsk / (frzsK + (1 - x(4)));
frzs_out_2 = m1 * x(4) * frzskd / (frzsKd + x(4));

mglaK = 1;
mglaKd = 1;

mglakd = 1;

d = m1;
```

```
mgla_in = d * three * (1 - x(5)) / (mglaK + (1 - x(5)));
mgla_out = mglakd * x(5) / (mglaKd + x(5));

% one x1 control inflow to 1
% two x2 control inflow to 2
% three frzs at 1
% four frzs at 2
% five - response to mgla signalling input

xprime = [(a1 / (b1 + x(2)^p1)) - (s1 * x(1));
    (a2 / (b2 + x(1)^p2)) - (s2 * x(2));
    (frzs_switch_2_1 * frzs_in_1) - (frzs_switch_1_2 * frzs_out_1);
    (frzs_switch_1_2 * frzs_in_2) - (frzs_switch_2_1 * frzs_out_2);
    mgla_in - mgla_out];
```

Appendix E

FABCell implementation of Wolfram rule 110 cellular automaton

This appendix lists a full implementation of the Wolfram 110 cellular automaton [194] in FABCell showing both the XML schema that describes the model and the C++ code required to start and run the simulation. The full model and all necessary code and resources is available as Model `mod_rule_110` on the `CD-ROM` accompanying this thesis (see Appendix F for details).

E.1 XML model definition

```
<environment id="0">
  <space type="plugin">
    <plugin name="space" lib="libfabcell_engine_space.so" call="SpacePluginCreate">
      <boundary>
        <x>none</x>
        <y>none</y>
        <z>cap</z>
      </boundary>
      <dimensions>
        <x>50</x>
        <y>50</y>
        <z>1</z>
      </dimensions>
    </plugin>
  </space>

  <mesh type="plugin">
```

```
  <plugin name="square_mesh" lib="libfabcell_plugins_square_mesh.so" call="SquareMeshPluginCreate">
    <neighboursCache use="true" capacity="200000" purge="100" />
    <boundary>
      <x>periodic</x>
      <y>periodic</y>
      <z>periodic</z>
    </boundary>
    <dimensions>
      <rows>auto</rows>
      <columns>auto</columns>
      <layers>auto</layers>
    </dimensions>

    <!-- describe linkages between nodes -->
    <channelOrientations>
      <directionCache>
        <directions>
          <direction type="square_3d">
            <reference>
              <coordinate x="1.0" y="0" z="0" />
            </reference>
          </direction>
        </directions>
      </directionCache>
    </channelOrientations>
  </plugin>
</mesh>

<occupants>
  <cellOrientations>
    <directionCache>
      <directions>
        <direction type="square_3d">
          <reference>
            <coordinate x="1.0" y="0" z="0" />
          </reference>
        </direction>
      </directions>
    </directionCache>
  </cellOrientations>
  <channels type="single" />
  <storage type="single" />
  <mapMode>normalise</mapMode>
```

```
</occupants>

<timer type="module">
  <name>standard</name>
  <start>0</start>
  <end>30</end>
  <increment>1</increment>
</timer>

<!-- every model needs at least one scheduler -->

<scheduler name="timed">
  <start>0</start>
  <interval>1</interval>

  <updates start="0" interval="1">
    <!--
    <update type="module">
      <name>frame_data_writer</name>
      <frameSkip>1</frameSkip>
      <prefix>frame_data_</prefix>
    </update>
    -->

    <!--
    <update type="module">
      <name>adm_data_writer</name>
      <calculateOrientation>false</calculateOrientation>
      <frameSkip>1</frameSkip>
      <prefix>adm_data_</prefix>
    </update>
    -->

    <update type="plugin">
      <plugin name="ca_writer" lib="libfabcell_plugins_ca.so" call="CaWriterPluginCreate">
        <frameSkip>1</frameSkip>
        <prefix>ca_data_</prefix>
      </plugin>
    </update>

    <!-- rule 110 from wolfram -->
    <update type="plugin">
      <plugin name="ca_rule" lib="libfabcell_plugins_ca.so" call="CaRulePluginCreate">
```

```
<type>1d_time</type>
<matches>
  <match type="plugin">
    <plugin name="specific" lib="libfabcell_plugins_ca.so"
    call="SpecificMatchPluginCreate">
      <currentState>black</currentState>
      <newState>white</newState>
      <neighbours>
        <neighbour row="0" column="-1" layer="0" state="black" />
        <neighbour row="0" column="1" layer="0" state="black" />
      </neighbours>
    </plugin>
  </match>
  <match type="plugin">
    <plugin name="specific" lib="libfabcell_plugins_ca.so"
    call="SpecificMatchPluginCreate">
      <currentState>black</currentState>
      <newState>black</newState>
      <neighbours>
        <neighbour row="0" column="-1" layer="0" state="black" />
        <neighbour row="0" column="1" layer="0" state="white" />
      </neighbours>
    </plugin>
  </match>
  <match type="plugin">
    <plugin name="specific" lib="libfabcell_plugins_ca.so"
    call="SpecificMatchPluginCreate">
      <currentState>white</currentState>
      <newState>black</newState>
      <neighbours>
        <neighbour row="0" column="-1" layer="0" state="black" />
        <neighbour row="0" column="1" layer="0" state="black" />
      </neighbours>
    </plugin>
  </match>
  <match type="plugin">
    <plugin name="specific" lib="libfabcell_plugins_ca.so"
    call="SpecificMatchPluginCreate">
      <currentState>white</currentState>
      <newState>white</newState>
      <neighbours>
        <neighbour row="0" column="-1" layer="0" state="black" />
        <neighbour row="0" column="1" layer="0" state="white" />
```

```
    </neighbours>
  </plugin>
</match>
<match type="plugin">
  <plugin name="specific" lib="libfabcell_plugins_ca.so"
  call="SpecificMatchPluginCreate">
    <currentState>black</currentState>
    <newState>black</newState>
    <neighbours>
      <neighbour row="0" column="-1" layer="0" state="white" />
      <neighbour row="0" column="1" layer="0" state="black" />
    </neighbours>
  </plugin>
</match>
<match type="plugin">
  <plugin name="specific" lib="libfabcell_plugins_ca.so"
  call="SpecificMatchPluginCreate">
    <currentState>black</currentState>
    <newState>black</newState>
    <neighbours>
      <neighbour row="0" column="-1" layer="0" state="white" />
      <neighbour row="0" column="1" layer="0" state="white" />
    </neighbours>
  </plugin>
</match>
<match type="plugin">
  <plugin name="specific" lib="libfabcell_plugins_ca.so"
  call="SpecificMatchPluginCreate">
    <currentState>white</currentState>
    <newState>black</newState>
    <neighbours>
      <neighbour row="0" column="-1" layer="0" state="white" />
      <neighbour row="0" column="1" layer="0" state="black" />
    </neighbours>
  </plugin>
</match>
<match type="plugin">
  <plugin name="specific" lib="libfabcell_plugins_ca.so"
  call="SpecificMatchPluginCreate">
    <currentState>white</currentState>
    <newState>white</newState>
    <neighbours>
      <neighbour row="0" column="-1" layer="0" state="white" />
```

```
                <neighbour row="0" column="1" layer="0" state="white" />
              </neighbours>
            </plugin>
          </match>
        </matches>
      </plugin>
    </update>

    <!--
    <update type="scheduler">
      <scheduler name="timed">
        <start>0</start>
        <interval>1</interval>

        <updates start="0" interval="0">
          <update type="module">
            <name>interaction_state_update</name>
          </update>
        </updates>
      </scheduler>
    </update>
    -->

    <update type="module">
      <name>cell_state_update</name>
    </update>
  </updates>
</scheduler>

<cells>
  <createCells name="all">
    <numberOfCells>all</numberOfCells>
    <segmentsPerCell>1</segmentsPerCell>
    <segmentSize>1</segmentSize>

    <cellCreator type="plugin">
      <plugin name="create_ca_cell" lib="libfabcell_plugins_ca.so"
      call="CreateCaCellPluginCreate">
        <initialState>white</initialState>
      </plugin>
    </cellCreator>

    <segmentsCreator type="standard" />
```

```
    <segmentCreator type="standard" />

    <segmentPopulator type="standard" />

    <segmentsPopulator type="standard">
      <checkOverlap>true</checkOverlap>
      <tolerance>0.5</tolerance>
      <neighbours type="moore">
        <distance>0</distance>
        <channels type="plugin">
          <plugin name="square_channels" lib="libfabcell_plugins_square_mesh.so"
           call="SquareChannelList2dPluginCreate" />
        </channels>
      </neighbours>
      <shift>
        <directionCache>
          <directions>
            <direction type="square_2d">
              <reference>
                <coordinate x="1.0" y="0" z="0" />
              </reference>
            </direction>
          </directions>
        </directionCache>
      </shift>
      <channels type="specific">
        <channel>0</channel>
        <!--<channel>2</channel>-->
      </channels>
    </segmentsPopulator>


    <startLocations type="grid">
      <rows>1</rows>
      <columns>1</columns>
      <layers>1</layers>
    </startLocations>
  </createCells>
</cells>

<!-- things to run to initialise the model -->
```

```
  <run>
    <runnable type="plugin">
      <plugin name="ca_initialise" lib="libfabcell_plugins_ca.so"
      call="CaInitialisePluginCreate">
        <nodes>
          <node>
            <address row="0" column="25" layer="0" />
            <state>black</state>
          </node>
        </nodes>
      </plugin>
    </runnable>

    <runnable type="plugin">
      <plugin name="cell_state_update" lib="libfabcell_engine_state.so"
      call="CellStateUpdatePluginCreate" />
    </runnable>

  </run>

</environment>
```

Appendix F

Instructions for compiling FABCell

This appendix provides instructions for compiling and running FABCell and the examples provided on the CD-ROM accompanying this thesis.

F.1 Obtaining the source code

The source code for FABCell is provided on the CD-ROM accompanying this thesis in the /code/fabcell/ directory. The most recent version is available from http://www.sourceforge.net/fabcell.

F.2 Compiling

FABCell is written in C++ on a GNU/Linux platform and as such should compile on any recent GNU/Linux distribution with the GCC/G++ libraries (greater than 3.4.6) installed. FABCell currently uses shared object libraries to manage its plug-in framework and cannot be compiled on a Microsoft® Windows® platform. Table F.1 lists the dependencies and libraries required to run FABCell:

1. Copy the source folder to a new directory and change to the new directory.
2. FABCell uses `make` to manage the build process.
3. Run `make clean` and `make all`.
4. To compile FABCell with OpenMP support, run `make all parallelbuild=1`.

Table F.1: FABCell dependencies.

Library	Notes
G++ 3.4.6 or later	G++ 4.0 is required for OpenMP support.
`libglut`	OpenGL Glut library.
`libGL`	
`libGLU`	
`libm`	
`libgd`	GD graphics library.
`libgsl`	GNU Scientific Library.
`libgslcblas`	
`libdl`	

5. Change to the `lib` directory and run `make copylib`. FABCell comprises multiple object libraries. This command will copy all of the libraries to the `lib` directory for ease of linking, otherwise each library will need to be manually linked when compiling a simulation to run.

To run a FABCell simulation, all of the libraries must be available in a location where the linker and executables can find them. If you do not have administrator rights on your system, use the `LD_LIBRARY_PATH` environmental variable to specify the folder where all the libraries are. For example:

```
LD_LIBRARY_PATH = libs

export $LD_LIBRARY_PATH
```

F.3 Running a model

Once the core FABCell modules have been compiled, the models can be compiled. All models are stored in:

```
/code/fabcell/examples/
```

Section F.5 details all of the available models and their respective folders. The compilation instructions for each model are the same.

Each model consists of the three primary files (see Table F.2): `model_executable`, `settings.xml` and `viewer.xml`. These files must be present when running a simulation. Other data files may be required by some of the models but these will be in the model directory. To compile the executable change to the directory for a particular

Table F.2: Files for FABCell model instantiation.

File	Notes
`model_executable`	The compiled executable to start and run a simulation.
`settings.xml`	An XML file describing the model and initialisation parameters.
`viewer.xml`	An XML file describing how simulation output should be rendered by the viewer.

model and type the following at the prompt:

```
make all
```

`make all` will create the executable which can then be run by typing the following at the shell prompt:

```
./model_executable
```

The `model_executable` will be different for each model. For example, in the case of Model `mod_rule_110` (see Section F.5), the executable is called `rule_110` and can be run as:

```
./rule_110
```

Depending on hardware and the complexity of the model, it may take several minutes to several hours to run a simulation to completion. The models can generate multiple gigabytes (GB) of data so it is recommended you have a large amount of free storage space available before running a simulation.

F.4 Visualising a model

FABCell has a three-dimensional viewer application for visualising model data. This can be found in:

`/code/fabcell/gui/viewer/viewer`

Copy `viewer` from the above directory to the directory where the model was run. Make sure `viewer.xml` is also in the directory. The model can be visualised by typing the following at the shell prompt:

```
./viewer
```

Some of the important keyboard controls and command line start up options are listed in Table F.3 and Table F.4 respectively. Command line arguments take the POSIX long form only and can be invoked as follows:

```
./viewer --option=value
```

The following example initialises the viewer to start at frame 10 rather than the first frame:

```
./viewer --frame=10
```

Table F.3: Keyboard controls for the FABCell viewer.

Key	Function
`n`	Move viewer to next frame.
`b`	Move viewer previous frame.
`r`	Reset to first frame.
`d`	Rotate viewer about the x-axis clockwise.
`c`	Rotate viewer about the x-axis anti-clockwise.
`a`	Rotate viewer about the y-axis clockwise.
`s`	Rotate viewer about the y-axis anti-clockwise.
`z`	Rotate viewer about the z-axis clockwise.
`x`	Rotate viewer about the z-axis anti-clockwise.
`q`	Zoom in.
`w`	Zoom out.
`o`	Save current viewer to image (`.png`).
`4`	Shift view to the left.
`6`	Shift view to the right.
`8`	Shift view upwards.
`2`	Shift view downwards.

Table F.4: Command line options for the FABCell viewer.

Argument	Function
`--frame=n`	Starts viewer at frame `n` rather than the first frame.
`--skip=n`	Viewer jumps `n` frames rather than one when keys `n` and `b` are used to alter the frame (see Table F.3).
`--rotx=n`	Starts viewer with the camera rotated an angle `n` from the default view around the x-axis.
`--roty=n`	Starts viewer with the camera rotated an angle `n` from the default view around the y-axis.
`--rotz=n`	Starts viewer with the camera rotated an angle `n` from the default view around the z-axis.
`--z=n`	Starts viewer with the camera a distance `n` from the model along the z-axis.

F.5 Example models

Table F.5 lists the location of the example models which are included on the CD-ROM. All of the models can be compiled and run using the instructions given in Section F.2 and Section F.3.

Table F.5: FABCell models

Model:	mod_rule_110
Name:	Wolfram Rule 110 CA
Directory:	/code/fabcell/examples/ca/rule_110/
An implementation of the Wolfram Rule 110 CA [194].	
Model:	mod_gol_gun
Name:	Game of Life Gun
Directory:	/code/fabcell/examples/gol/gun/
An CA implementation of the Game Of Life demonstrating a *gun*[1].	
Model:	mod_gol_qbs
Name:	Game of Life Queen Bee Shuttle
Directory:	/code/fabcell/examples/gol/qbs/
A CA implementation of the Game Of Life demonstrating the *queen bee shuttle*.	
Model:	mod_cell_sort
Name:	CPMcell sorting
Directory:	/code/fabcell/examples/cell_sort/
A CPM implementation of the cell sorting model described by Graner and Glazier [49].	
Model:	mod_mot
Name:	*Motilator*
Directory:	/code/fabcell/examples/motilator/
An LGCA model of myxobacteria cell rippling using the *Motilator* as described in Chapter 6.	
Model:	mod_mot_no_coll
Name:	*Motilator*(no collisions)
Directory:	/code/fabcell/examples/motilator/

An LGCA model of myxobacteria cell rippling without collisions using the *Motilator* as described in Chapter 6.

Model: `mod_phys_mot`

Name: *Phys-Motilator*

Directory: `/code/fabcell/examples/phys_motilator/`

A combined model of myxobacteria cell rippling using the *Motilator* and a Monte Carlo model of cell dynamics as described in Chapter 7.

Model: `mod_fruit`

Name: *Fruit-Motilator*

Directory: `/code/fabcell/examples/fruit_motilator/`

A Monte Carlo model of fruiting body development as described in Chapter 8.

Bibliography

[1] M.S. Alber, D.N. Arnold, M.A. Kiskowski, F. Santosa, J.A. Glazier, and Y. Jiang. On cellular automaton approaches to modeling biological cells. In J. Rosenthal and D.S. Gilliam, editors, *IMA 134: mathematical systems theory in biology, communication, and finance*, pages 1–40. Springer-Verlag, 2003.

[2] M.S. Alber, Y. Jiang, and M.A. Kiskowski. Lattice gas cellular automation model for rippling and aggregation in myxobacteria. *Physica D*, 191:343–358, 2003.

[3] M.S. Alber, M.A. Kiskowski, and Y. Jiang. Two-stage aggregate formation via streams in myxobacteria. *Phys. Rev. Lett.*, 93:1–4, 2004.

[4] M.S. Alber, M.A. Kiskowski, Y. Jiang, and S. Newman. Biological lattice gas models. In G. Dangelmayr and I. Oprea, editors, *Dynamics and bifurcation of patterns in dissipative systems*, pages 274–291. World Scientific Publishing, 2004.

[5] R.A. Anderson and B.N. Vasiev. An individual based model of rippling movement in a myxobacteria population. *J. Theor. Biol.*, 234:341–349, 2005.

[6] D. Angeli, J.E. Ferrell, and E.D. Sontag. Detection of multistability, bifurcations, and hysteresis in a large class of biological positive-feedback systems. *Proc. Natl. Acad. Sci. U.S.A.*, 101:1822–1827, 2004.

[7] J.W. Arnold and L.J. Shimkets. Cell surface properties correlated with cohesion in *Myxococcus xanthus*. *J. Bacteriol.*, 170:5771–5777, 1988.

[8] F.M. Ausubel, R. Brent, R.E. Kingston, D. Moore, J.G. Seidman, A. Smith, and K. Struhl, editors. *Short protocols in molecular biology, fifth edition*. Wiley, 2002.

[9] B. Barney. Posix threads programming. `https://computing.llnl.gov/tutorials/pthreads/`, May 2009.

[10] J.E. Berleman and J.R. Kirby. Multicellular development in *Myxococcus xanthus* is stimulated by predator-prey interactions. *J. Bacteriol.*, 2007.

[11] J.E. Berleman, T. Chumley, P. Cheung, and J.R. Kirby. Rippling is a predatory behavior in *Myxococcus xanthus*. *J. Bacteriol.*, pages 5888–5895, 2006.

[12] J.E. Berleman, J. Scott, T. Chumley, and J.R. Kirby. Predataxis behavior in *Myxococcus xanthus*. *Proc. Natl. Acad. Sci. U.S.A.*, 105:17127–17132, 2008.

[13] J. Bishop. *C# 3.0 design patterns*, chapter 1: C# meets design patterns. O'Reilly, 2008.

[14] J. Bishop. *C# 3.0 design patterns*, chapter 2: structural patterns: decorator, proxy, bridge. O'Reilly, 2008.

[15] J. Bishop. *C# 3.0 design patterns*, chapter 9: Behavioral Patterns: iterator, mediator, and observer. O'Reilly, 2008.

[16] J. Bishop. *C# 3.0 design patterns*, chapter 10: Behavioral patterns: visitor, interpreter, and memento. O'Reilly, 2008.

[17] J. Bishop. *C# 3.0 design patterns*, chapter 5: creational patterns: prototypes, factory method and singleton. O'Reilly, 2008.

[18] B.D. Blackhart and D.R. Zusman. Frizzy genes of *Myxococcus xanthus* are involved in control of frequency of reversal of gliding motility. *Proc. Natl. Acad. Sci. U.S.A.*, 88:8767–8770, 1985.

[19] OpenMP Architecture Review Board. OpenMP.org. `http://openmp.org/wp/`, May 2009.

[20] U. Börner, A. Deutsch, H. Reichenbach, and M. Bär. Rippling patterns in aggregates of myxobacteria arise from cell-cell collisions. *Physical Review*, 89:78100–78097, 2002.

[21] U. Börner, A. Deutsch, and M. Bär. A generalised discrete model linking rippling pattern formation and individual cell reversal statistics in colonies of myxobacteria. *Phys. Biol.*, 3:138–146, 2006.

[22] A.P. Bretscher and D. Kaiser. Nutrition of *Myxococcus xanthus*, a fruiting myxobacterium. *J. Bacteriol.*, 133:763–768, 1978.

[23] R.P. Burchard. Trail following by gliding bacteria. *J. Bacteriol.*, 152:495–501, 1982.

[24] V.H. Bustamente, I. Martinez-Flores, H.C. Vlamakis, and D.R. Zusman. Analysis of the frz signal transduction system of *Myxococcus xanthus* shows the importance of the conserved C-terminal region of the cytoplasmic chemoreceptor FrzCD in sensing signals. *Mol. Microbiol.*, 53:1501–1513, 2004.

[25] S.L. Campbell and R. Haberman. *Introduction to differential equations with dynamical systems*. Princeton University Press, 2008.

[26] J.M. Campos and D.R. Zusman. Regulation of development in *Myxococcus xanthus*: effect of 3':5'-cyclic AMP, ADP, and nutrition. *Proc. Natl. Acad. Sci. U.S.A.*, 72:518–522, 1975.

[27] S. Chib and E. Greenberg. Understanding the Metropolis-Hastings algorithm. *American Statistician*, 49:327–335, 1995.

[28] T.M. Cickovski, C. Huang, R. Chaturvedi, T.H. Glimm, G.E. Hentschel, M.S. Alber, J.A. Glazier, S.A. Newman, and J.A. Izaguirre. A framework for three-dimensional simulation of morphogenesis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2:1–16, 2005.

[29] P. Cousot. Methods and logics for proving programs. In J. Van Leeuwen, editor, *Handbook of theoretical computer science: volume B*, pages 843–993. Elsevier Science Publishers, 1990.

[30] P.D. Curtis, R. Geyer, D.C. White, and L.J. Shimkets. Novel lipids in *Myxococcus xanthus* and their role in chemotaxis. *Environ. Microbiol.*, 8:1935–1949, 2006.

[31] P.D. Curtis, R.G. Taylor, R.D. Welch, and L.J. Shimkets. Spatial organization of *Myxococcus xanthus* during fruiting body formation. *J. Bacteriol.*, 189:9126–9130, 2007.

[32] A. Deutsch and S. Dormann. *Cellular automaton: modelling of biological pattern formation*. Birkhauser, 2005.

[33] E.W. Dijkstra. Cooperating sequential processes. Technical report, Communications of the ACM, 1965.

[34] E.W. Dijkstra. Guarded commands, non-determinacy and formal derivation of programs. *Communications of the ACM*, 8:453–457, 1975.

[35] M.E. Diodati, R.E. Gill, L. Plamann, and M. Singer. Initiation and early developmental events. In D.E. Whitworth, editor, *Myxobacteria: multicellularity and differentiation*, pages 43–76. American Society for Microbiology, 2008.

[36] M. Dworkin. Nutritional requirements for vegetative growth of *Myxococcus xanthus*. *J. Bacteriol.*, 84:250–257, 1962.

[37] M. Dworkin. Recent advances in the social and developmental biology of the Myxobacteria. *Microbiol. Rev.*, 60:70–102, 1996.

[38] T. Emonet, C.M. Macal, M.J. North, C.E. Wickersham, and P. Cluzel. Agentcell: a digital single-cell assay for bacterial chemotaxis. *Bioinformatics*, 21:2714–2721, 2005.

[39] J.J. Falke, R.B. Bass, S.L. Butler, S.A. Chervitz, and M.A. Danielson. The two-component signaling pathway of bacterial chemotaxis: a molecular view of signal transduction by receptors, kinases, and adaptation enzymes. *Annu. Rev. Cell Dev. Biol.*, 13:457–512, 1997.

[40] F. Fiegna and G.J. Velicer. Competitive fates of bacterial social parasites: persistence and self-induced extinction of *Myxococcus xanthus* cheaters. *The Royal Society*, 270:1527–1534, 2003.

[41] Free Software Foundation. Gnu scientific library. `http://www.gnu.org/software/gsl/`, August 2009.

[42] Free Software Foundation. The gnu compiler collection. `http://gcc.gnu.org/`, August 2009.

[43] M. Gardner. Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, 223:120–123, 1970.

[44] T.S. Gardner, C.R. Cantor, and J.J. Collins. Construction of a genetic toggle switch in *Escherichia coli*. *Nature*, 403:339–342, 2000.

[45] R.E. Gill and M.C. Bornemann. Identification and characterization of the *Myxococcus xanthus bsgA* gene product. *J. Bacteriol.*, 170:5289–5297, 1988.

[46] R.E. Gill, M. Karlok, and D. Benton. *Myxococcus xanthus* encodes an ATP-dependent protease which is required for developmental gene transcription and intercellular signaling. *J. Bacteriol.*, 175:4538–4544, 1993.

[47] J.A. Glazier and F. Graner. Simulation of the differential adhesion driven rearrangement of biological cells. *Physical Review E*, 47:2128–2154, 1993.

[48] B.S. Goldman, W.C. Nierman, D. Kaiser, S.C. Slater, A.S. Durkin, J. Eisen, C.M. Ronning, W.B. Barbazuk, M. Blanchard, C. Field, C. Halling, G. Hinkle, O. Iartchuk, H.S. Kim, C. Mackenzie, R. Madupu, N. Miller, A. Shvartsbeyn, S.A. Sullivan, M. Vaudin, R. Wiegand, and H.B. Kaplan. Evolution of sensory complexity recorded in a myxobacterial genome. *Proc. Natl. Acad. Sci. U.S.A.*, 103:15200–15205, 2006.

[49] F. Graner and J.A. Glazier. Simulation of biological cell sorting using a two-dimensional extended potts model. *Phys. Rev. Lett.*, 69:2013–2016, 1992.

[50] P.L. Grilione and J. Pangborn. Scanning electron microscopy of fruiting body formation by myxobacteria. *J. Bacteriol.*, 124:1558–1565, 1975.

[51] Swarm Development Group. SwarmWiki: tools for agent-based modelling. `http://www.swarm.org/`, May 2009.

[52] J.W. Haefner. *Modeling biological systems: principles and applications*, chapter 1: Uses of scientific models. Chapman and Hall, 1996.

[53] J.W. Haefner. *Modeling biological systems: principles and applications*, chapter 2: the modeling process. Chapman and Hall, 1996.

[54] T.J. Hagen and L.J. Shimkets. Nucleotide sequence and transcriptional products of the *csg* locus of *Myxococcus xanthus*. *J. Bacteriol.*, 172:15–23, 1990.

[55] E.R. Harold and W.S. Means. *XML in a nutshell, third edition*, chapter 1: XML concepts. O'Reilly, 2004.

[56] J. Hasty, D. McMillen, and J.J. Collins. Engineered gene circuits. *Nature*, 420: 224–230, 2002.

[57] M. Hendrata and B. Birnir. The dynamics of myxobacteria life cycle. Technical report, Center for Complex and Nonlinear Science, 2008.

[58] J. Hodgkin and D. Kaiser. Genetics of gliding motility in *Myxococcus xanthus* (myxobacterales): Two gene systems control movement. *Mol. Gen. Genet.*, 171: 177–191, 1979.

[59] A.B. Holmes. FABCell. `http://sourceforge.net/projects/fabcell/`, May 2009.

[60] A.B. Holmes, S. Kalvala, and D.E. Whitworth. Myxobacteria motility: a novel 3D model of rippling behaviour in *Myxococcus xanthus*. *Communications of the Systematics and Informatics World Network*, 6:65–70, 2009.

[61] O.A. Igoshin and G. Oster. Rippling of myxobacteria. *Mathematical Biosciences*, 188:221–233, 2004.

[62] O.A. Igoshin, A. Mogilner, R.D. Welch, and D. Kaiser. Pattern formation and traveling waves in myxobacteria: theory and modeling. *Biophysics*, 98:14913–14918, 2001.

[63] O.A. Igoshin, A. Goldbeter, D. Kaiser, and G. Oster. A biochemical oscillator explains several aspects of *Myxococcus xanthus* behavior during development. *Proc. Natl. Acad. Sci. U.S.A.*, 101:15760–15765, 2004.

[64] O.A. Igoshin, D. Kaiser, and G. Oster. Breaking symmetry in myxobacteria. *Curr. Biol.*, 14:R459–R462, 2004.

[65] O.A. Igoshin, R.D. Welch, D. Kaiser, and G. Oster. Waves and aggregation patterns in myxobacteria. *Proc. Natl. Acad. Sci. U.S.A.*, 101:4256–4261, 2004.

[66] T. Iizukaa, Y. Jojimaa, R. Fudoua, and S. Yamanakaa. Isolation of myxobacteria from the marine environment. *FEMS Microbiol. Ecol.*, 169:317–322, 1998.

[67] S. Ishihara, K. Fujimoto, and T. Shibata. Cross talking of network motifs in gene regulation that generates temporal pulses and spatial stripes. *Genes Cells*, 10: 1025–1038, 2005.

[68] Telecom Italia. Jade (java agent development framework). `http://jade.cselt.it/`, May 2009.

[69] J.A. Izaguirre, R. Chaturvedi, C. Huang, T.M. Cickovski, J. Coffland, G. Thomas, G. Forgacs, M.S. Alber, G.E. Hentschel, S.A. Newman, and J.A. Glazier. Compucell, a multi-model framework for simulation of morphogenesis. *Bioinformatics*, 20:1129–1137, 2003.

[70] L. Jelsbak and L. Søgaard-Anderson. The cell surface-associated intercellular c-signal induces behavioral changes in individual *Myxococcus xanthus* cells during fruiting body morphogenesis. *Proc. Natl. Acad. Sci. U.S.A.*, 96:5031–5036, 1999.

[71] L. Jelsbak and L. Søgaard-Anderson. Pattern formation by a cell surface-associated morphogen in *Myxococcus xanthus*. *Proc. Natl. Acad. Sci. U.S.A.*, 99: 2032–2037, 2002.

[72] L. Jelsbak and L. Søgaard-Anderson. Cell behavior and cell-cell communication during fruiting body morphogenesis in *Myxococcus xanthus*. *J. Microbiol. Methods*, 55:829–839, 2003.

[73] N.R. Jennings. On agent-based software engineering. *Artificial Intelligence Journal*, 117:277–296, 2000.

[74] D. Kaiser. Social gliding is correlated with the presence of pili in *Myxococcus xanthus*. *Proc. Natl. Acad. Sci. U.S.A.*, 76:5952–5956, 1979.

[75] D. Kaiser. Roland Thaxter's legacy and the origins of multicellular development. *Genetics*, 135:249–254, 1993.

[76] D. Kaiser. Signaling in myxobacteria. *Annu. Rev. Microbiol.*, 58:75–98, 2004.

[77] D. Kaiser. Reversing *Myxococcus xanthus* polarity. In D.E. Whitworth, editor, *Myxobacteria: multicellularity and differentiation*, pages 93–122. American Society for Microbiology, 2008.

[78] D. Kaiser and M. Dworkin. From glycerol to the genome. In D.E. Whitworth, editor, *Myxobacteria: multicellularity and differentiation*, pages 3–15. American Society for Microbiology, 2008.

[79] D. Kaiser and L. Kroos. Intercellular signaling. In M. Dworkin and D. Kaiser, editors, *Myxobacteria II*, pages 257–283. American Society for Microbiology, 1993.

[80] D. Kaiser and R. Losick. How and why bacteria talk to each other. *Cell*, 73: 873–885, 1993.

[81] D. Kaiser and R.D. Welch. Dynamics of fruiting body morphogenesis. *J. Bacteriol.*, 186:919–927, 2004.

[82] D. Kaiser, C. Manoil, and M. Dworkin. Myxobacteria: cell interactions, genetics and development. *Annu. Rev. Microbiol.*, 33:595–639, 1979.

[83] G. Kaur, S. Holbeck, V. Schauer-Vukasinovic, R.F. Camalier, A. Domling, and S Agarwal. Biological evaluation of tubulysin a: a potential anticancer and anti-angiogenic natural product. *Biochem. J.*, 396:235–242, 2006.

[84] S.K. Kim and D. Kaiser. C-factor: A cell-cell signaling protein required for fruiting body morphogenesis of *M. Xanthus*. *Cell*, 61:19–26, 1990.

[85] S.K. Kim and D. Kaiser. Cell alignment required in differentiation of *Myxococcus xanthus*. *Science*, 249:926–928, 1990.

[86] S.K. Kim and D. Kaiser. C-factor has distinct aggregation and sporulation thresholds during *Myxococcus* development. *J. Bacteriol.*, 173:1722–1728, 1991.

[87] M.A. Kiskowski, Y. Jiang, and M.S. Alber. Role of streams in myxobacteria aggregate formation. *Phys. Biol.*, 1:173–183, 2004.

[88] H. Kobayashi, M. Kaern, M. Araki, K. Chung, T.S. Gardner, C.R. Cantor, and J.J. Collins. Programmable cells: Interfacing natural and engineered gene networks. *Proc. Natl. Acad. Sci. U.S.A.*, 101:8414–8419, 2004.

[89] T. Kruse, S. Lobedanz, N.M.S Berthelsen, and L. Søgaard-Anderson. C-signal: a cell surface-associated morphogen that induces and co-ordinates multicellular fruiting body morphogenesis and sporulation in *Myxococcus xanthus*. *Mol. Microbiol.*, 40:156–168, 2001.

[90] J.M. Kuner and D. Kaiser. Fruiting body morphogenesis in submerged cultures of *Myxococcus xanthus*. *J. Bacteriol.*, 151:458–461, 1982.

[91] Argonne National Laboratory. Repast agent simulation toolkit. `http://repast.sourceforge.net/`, May 2009.

[92] B.E. Laue and R.E. Gill. Use of a phase variation-specific promoter of *Myxococcus xanthus* in a strategy for isolating a phase-locked mutant. *J. Bacteriol.*, 176:5341–5349, 1994.

[93] S. Leonardy, G. Freymark, S. Hebener, E. Ellehauge, and L. Søgaard-Anderson. Coupling of protein localization and cell movements by a dynamically localized response regulator in *Myxococcus xanthus*. *EMBO J.*, 26:4433–4444, 2007.

[94] S. Leonardy, I. Bulyha, and L. Søgaard-Anderson. Reversing cells and oscillating motility proteins. *Mol. BioSyst.*, 4:1009–1014, 2008.

[95] Canonical Ltd. Ubuntu. `http://www.ubuntu.com/`, August 2009.

[96] S. Luke, C. Cioffi-Revilla, L. Panait, and K. Sullivan. MASON: a new multi-agent simulation toolkit. *Proceedings of the 2004 SwarmFest Workshop*, 2004.

[97] C. Manoil and D. Kaiser. Guanosine pentaphosphate and guanosine tetraphosphate accumulation and induction of myxococcus xanthus fruiting body development. *J. Bacteriol.*, 141:305–315, 1980.

[98] N.L. Markevich, J.B. Hoek, and B.N. Kholodenko. Signaling switches and bistability arising from multisite phosphorylation in protein kinase cascades. *J. Cell Biol.*, 164:353–359, 2004.

[99] E.M.F. Mauriello, D.P. Astling, O. Sliusarenko, and D.R. Zusman. Localization of a bacterial cytoplasmic receptor is dynamic and changes with cell-cell contacts. *Proc. Natl. Acad. Sci. U.S.A.*, 2009.

[100] E.M.F. Mauriello, B. Nan, and D.R. Zusman. AglZ regulates adventurous (A-) motility in emphMyxococcus xanthus through its interaction with the cytoplasmic receptor, FrzCD. *Mol. Microbiol.*, 2009.

[101] M.J. McBride. Bacterial gliding motility: multiple mechanisms for cell movement over surfaces. *Annu. Rev. Microbiol.*, 55:49–75, 2001.

[102] M.J. McBride and D.R. Zusman. FrzCD, a methyl-accepting taxis protein from *Myxococcus xanthus*, shows modulated methylation during fruiting body formation. *J. Bacteriol.*, 175:4936–4940, 1993.

[103] W.R. McCleary, M.J. McBride, and D.R. Zusman. Developmental sensory transduction in *Myxococcus xanthus* involves methylation and demethylation of FrzCD. *J. Bacteriol.*, 172:4877–4887, 1990.

[104] A. McVittie, F. Messik, and S.A. Zahler. Developmental biology of *Myxococcus*. *J. Bacteriol.*, 84:546–551, 1962.

[105] N. Metropolis. The beginning of the Monte Carlo method. *Los Alamos Science*, 15:125–130, 1987.

[106] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, and A.H. Teller. Equations of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087–1092, 1953.

[107] Sun Microsystems. java.com. `http://www.java.com/en/`, May 2009.

[108] T. Mignot. The elusive engine in *Myxococcus xanthus* gliding motility. *Cell. Mol. Life Sci.*, pages 1–13, 2007.

[109] T. Mignot and J.R. Kirby. Genetic circuitry controlling motility behaviors of *Myxococcus xanthus*. *BioEssays*, 30:733–743, 2008.

[110] T. Mignot, J.P. Merlie, and D.R. Zusman. Regulated pole-to-pole oscillations of a bacterial gliding motility protein. *Science*, 310:855–857, 2005.

[111] T. Mignot, J.P. Merlie, and D.R. Zusman. Two localization motifs mediate polar residence of frzs during cell movement and reversals of *Myxococcus xanthus*. *Mol. Microbiol.*, 65:363–372, 2007.

[112] T. Mignot, J.W. Shaevitz, P.L. Hartzell, and D.R. Zusman. Evidence that focal adhesion complexes power bacterial gliding motility. *Science*, 315:853–856, 2007.

[113] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[114] A. Moraleda-Munoz, J. Carrero-Lerida, A. Extremera, J.M. Arias, and J. Munoz-Dorado. Glycerol 3-phosphate inhibits swarming and aggregation of *Myxococcus xanthus*. *J. Bacteriol.*, 183:6135–6139, 2001.

[115] M.E.J. Newman and G.T. Barkema. *Monte Carlo methods in statistical physics*, chapter The Isling model and the Metropolis algorithm. Oxford University Press, 2004.

[116] M.E.J. Newman and G.T. Barkema. *Monte Carlo methods in statistical physics*. Oxford University Press, 2004.

[117] L. Null and J. Lobur. *The essentials of computer organization and architecture, second edition*. Jones and Bartlett Publishers, 2006.

[118] K.A. O'Connor and D.R. Zusman. Patterns of cellular interactions during fruiting-body formation in *Myxococcus xanthus*. *J. Bacteriol.*, 171:6013–6024, 1989.

[119] K.A. O'Connor and D.R. Zusman. Behavior of peripheral rods and their role in the life cycle of *Myxococcus xanthus*. *J. Bacteriol.*, 173:3342–3355, 1991.

[120] K.A. O'Connor and D.R. Zusman. Development in *Myxococcus xanthus* involves differentiation into two cell types, peripheral rods and spores. *J. Bacteriol.*, 173: 3318–3333, 1991.

[121] Persistence of Vision Raytracer Pty. Ltd. POV-Ray: the persistence of vision raytracer.
`http://www.povray.org/`, May 2009.

[122] K. Ollivier. wxwidgets.
`http://www.wxwidgets.org/`, May 2009.

[123] OpenOffice.org. OpenOffice.org: the free and open productivity suite.
`http://www.openoffice.org/`, May 2009.

[124] E. Palsson and H.G. Othmer. A model for individual and collective cell movement in *Dictyostelium discoideum*. *Proc. Natl. Acad. Sci. U.S.A.*, 97:10448–10453, 2000.

[125] D.C. Parker, S.M. Manson, M.A. Janssen, M.J. Hoffmann, and P. Deadman. Multi-agent systems for the simulation of land-use and land-cover change: a review. *Annals of the Association of American Geographers*, 93:314–337, 2003.

[126] A.E. Pelling, Y. Li, S.E. Cross, S Castaneda, W. Shi, and J.K. Gimzewski. *Self-organized and highly ordered domain structures within swarms of* Myxococcus xanthus, volume 63. Cell Motil. Cytoskeleton, 2006.

[127] V.D. Pham, C.W. Shebelut, M.E. Diodati, C.T. Bull, and M. Singer. Mutations affecting predation ability of the soil bacterium *Myxococcus xanthus*. *Microbiology*, 151:1865–1874, 2005.

[128] R.B. Potts. Some generalized order-disorder transformations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 48:106–109, 1952.

[129] M. Priestley. *Practical object-oriented design with UML, second edition*. McGraw-Hill, 2003.

[130] The CellML project. The cellml project. `http://www.cellml.org/`, August 2009.

[131] The Open MPI Project. Open MPI: open source high performance computing. `http://www.open-mpi.org/`, May 2009.

[132] R. Reed, D. Holmes, J. Weyers, and A. Jones. *Practical skills in biomolecular sciences, third edition*. Pearson Education, 2007.

[133] H. Reichenbach. Myxobacteria: a most peculiar group of social prokaryotes. In E. Rosenberg, editor, *Myxobacteria: development and cell interactions*, pages 1–50. Springer-Verlag, 1984.

[134] H. Reichenbach. Biology of the myxobacteria: ecology and taxonomy. In M. Dworkin and D. Kaiser, editors, *Myxobacteria II*, pages 13–62. American Society for Microbiology, 1993.

[135] H. Reichenbach and G. Höfle. Discovery and development of the epothilones : a novel class of antineoplastic drugs. *Drugs R D*, 9:1–10, 2008.

[136] H. Reichenbach, H. Voelz, and M. Dworkin. Structural changes in *Stigmatella aurantiaca* during myxospore induction. *J. Bacteriol.*, 97:905–911, 1969.

[137] C. Rodriguez-Navarro, M. Rodriguez-Gallego, K.B. Chekroun, and M.T. Gonzalez-Munoz. Conservation of ornamental stone by *Myxococcus xanthus*-induced carbonate biomineralization. *Appl. Environ. Microbiol.*, 69:2182–2193, 2003.

[138] E. Rosenberg, editor. *Myxobacteria: development and cell interactions*. Springer-Verlag, 1984.

[139] E. Rosenberg and M. Varon. Antibiotics and lytic enzymes. In E. Rosenberg, editor, *Myxobacteria: development and cell interactions*, pages 109–125. Springer-Verlag, 1984.

[140] E. Rosenberg, K.H. Keller, and M. Dworkin. Cell density-dependent growth of *Myxococcus xanthus* on casein. *J. Bacteriol.*, 129:770–777, 1977.

[141] B. Sager and D. Kaiser. Intercellular C-signaling and the traveling waves of *Myxococcus*. *Genes Dev.*, 8:2793–2804, 1994.

[142] N.J. Savill and P. Hogeweg. Modelling morphogenesis: from single cells to crawling slugs. *J. Theor. Biol.*, 184:229–235, 1996.

[143] SBML.org. Systems biology markup language. `http://sbml.org/`, August 2009.

[144] C. Schenk. MiKTeX project page. `http://miktex.org/`, May 2009.

[145] H. Schildt. *C++: the complete reference, fourth edition*. McGraw-Hill, 2002.

[146] J.A. Shapiro. Thinking about bacterial populations as multicellular organisms. *Annu. Rev. Microbiol.*, 52:81–104, 1998.

[147] W. Shi and D.R. Zusman. The two motility systems of *Myxococcus xanthus* show different selective advantages on various surfaces. *Proc. Natl. Acad. Sci. U.S.A.*, 90:3378–3382, 1993.

[148] W. Shi, F.K. Ngok, and D.R. Zusman. Cell density regulates cellular reversal frequency in *Myxococcus xanthus*. *Proc. Natl. Acad. Sci. U.S.A.*, 93:4142–4146, 1996.

[149] L.J. Shimkets. Intercellular signaling during fruiting-body development of *Myxococcus xanthus*. *Annu. Rev. Microbiol.*, 53:525–549, 1999.

[150] L.J. Shimkets and D. Kaiser. Induction of coordinated movement of *Myxococcus xanthus* cells. *J. Bacteriol.*, 152:451–461, 1982.

[151] L.J. Shimkets and H. Rafiee. CsgA, an extracellular protein essential for *Myxcoccus xanthus* development. *Proc. Natl. Acad. Sci. U.S.A.*, 172:5299–5306, 1990.

[152] L.J. Shimkets and T.W. Seale. Fruiting-body formation and myxospore differentiation and germination in *Myxococcus xanthus* viewed by scanning electron microscopy. *J. Bacteriol.*, 121:711–720, 1975.

[153] L.J. Shimkets, M. Dworkin, and H. Reichenbach. The myxobacteria. In M. Dworkin, editor, *The Prokaryotes*, volume 7. Springer, 2006.

[154] O. Sliusarenko, J. Neu, D.R. Zusman, and G. Oster. Accordion waves in *Myxococcus xanthus*. *Proc. Natl. Acad. Sci. U.S.A.*, 103:1534–1539, 2006.

[155] O. Sliusarenko, D.R. Zusman, and G. Oster. Aggregation during fruiting body formation in *Myxococcus xanthus* is driven by reducing cell movement. *J. Bacteriol.*, 189:611–619, 2007.

[156] L. Søgaard-Anderson. Cell polarity, intercellular signalling and morphogenetic cell movements in *Myxococcus xanthus*. *Curr. Opin. Microbiol.*, 7:587–593, 2004.

[157] L. Søgaard-Anderson and D. Kaiser. C factor, a cell-surface-associated intercellular signaling protein, stimulates the cytoplasmic Frz signal transduction system in *Myxococcus xanthus*. *Proc. Natl. Acad. Sci. U.S.A.*, 93:2675–2679, 1996.

[158] I. Sommerville. *Software engineering, seventh edition*. Pearson Education, 2004.

[159] I. Sommerville. *Software engineering, seventh edition*, chapter 23: Software testing. Pearson Education, 2004.

[160] O. Sozinova, Y. Jiang, D. Kaiser, and M.S. Alber. A three-dimensional model of myxobacterial aggregation by contact-mediated interactions. *Proc. Natl. Acad. Sci. U.S.A.*, 102:11308–11312, 2005.

[161] O. Sozinova, Y. Jiang, D. Kaiser, and M.S. Alber. A three-dimensional model of myxobacterial fruiting-body formation. *Proc. Natl. Acad. Sci. U.S.A.*, 103:17255–17259, 2006.

[162] A.M. Spormann. Gliding motility in bacteria: Insights from studies of *Myxococcus xanthus*. *Microbiol. Mol. Biol. Rev.*, 63:621–641, 1999.

[163] A.M. Spormann and D. Kaiser. Gliding movements in *Myxococcus xanthus*. *J. Bacteriol.*, 177:5846–5852, 1995.

[164] A.M. Spormann and D. Kaiser. Gliding mutants of *Myxococcus xanthus* with high reversal frequencies and small displacements. *J. Bacteriol.*, 181:2593–2601, 1999.

[165] J. Starruss, T. Bley, L. Søgaard-Anderson, and A. Deutsch. A new mechanism for collective migration in *Myxococcus xanthus*. *J. Stat. Phys.*, 128:269–286, 2007.

[166] U. Stern. Java vs. c++.
`http://verify.stanford.edu/uli/java_cpp.html`, May 2009.

[167] A. Stevens. Trail following and aggregation of myxobacteria. *J. Biol. Systems*, 3: 1059–1068, 1995.

[168] A. Stevens. A stochastic cellular automaton modelling gliding and aggregation of myxobacteria. *SIAM J. Appl. Math.*, 61:172–182, 2000.

[169] A. Stevens and L. Søgaard-Anderson. Making waves: pattern formation by a cell-surface-associated signal. *Trends Microbiol.*, 13:249–252, 2005.

[170] M. Sugimoto, K. Takahashi, T. Kitayama, D. Ito, and M. Tomita. Distributed cell biology simulations with E-Cell system. *Springer-Verlag*, pages 20–31, 2005.

[171] R.J. Swift and S.A. Wirkus. *A course in ordinary differential equations*. Chapman and Hall/CRC, 2007.

[172] K. Takahashi, K. Yugi, K. Hashimoto, Y. Yamada, C.J.F. Pickett, and M. Tomita. Computational challenges in cell simulation: a software engineering approach. *IEEE Intelligent Systems*, pages 64–71, 2002.

[173] K. Takahashi, K. Kaizu, B. Hu, and M. Tomita. A multi-algorithm, multi-timescale method for cell simulation. *Bioinformatics*, 20:538–546, 2004.

[174] TeXnicCenter.org. TeXnicCenter. `http://www.texniccenter.org/`, May 2009.

[175] R. Thaxter. Further observations on the myxobacteriaceae. *Boz. Gaz.*, 23:395–411, 1897.

[176] M. Tomita, K. Hashimoto, K. Takahashi, T.S. Shimizu, Y. Matsuzaki, F. Miyoshi, K. Saito, S. Tanida, K. Yugi, J.C. Venter, and C.A. Hutchison. E-Cell: software environment for whole cell simulation. *Bioinformatics*, 15:72–84, 1999.

[177] M. Travisano and G.J. Velicer. Strategies of microbial cheater control. *Trends Microbiol.*, 12:72–78, 2004.

[178] H. Van Vliet. *Sofware Engineering: principles and practice, third edition*. John Wiley and Sons, 2008.

[179] H. Van Vliet. *Sofware Engineering: principles and practice, third edition*, chapter 9: requirements engineering. John Wiley and Sons, 2008.

[180] H. Van Vliet. *Sofware Engineering: principles and practice, third edition*, chapter 15: software tools. John Wiley and Sons, 2008.

[181] H. Van Vliet. *Sofware Engineering: principles and practice, third edition*, chapter 13: Software testing. John Wiley and Sons, 2008.

[182] H. Voelz. The fate of the cell envelopes of *Myxococcus xanthus* during microcyst germination. *Arch. Microbiol.*, 55:110–115, 1966.

[183] H. Voelz and M. Dworkin. Fine structures of *Myxococcus xanthus* during morphogenesis. *J. Bacteriol.*, 84:943–952, 1962.

[184] J. von Neumann and A.W. Burks, editors. *Theory of self-reproducing automata*. U. of Illinois Press, 1966.

[185] B.F. Watson and M. Dworkin. Comparative intermediary metabolism of vegetative cells and microcysts of *Myxococcus xanthus*. *J. Bacteriol.*, 96:1465–1473, 1968.

[186] A. Webster and E. May. Bioremediation of weathered-building stone surfaces. *Trends Microbiol.*, 24:255–260, 2006.

[187] J. Weijer. Dictyostelium morphogenesis. *Curr. Opin. Genet. Dev.*, 14:392–398, 2004.

[188] R.D. Welch and D. Kaiser. Cell behavior in traveling wave patterns of myxobacteria. *Proc. Natl. Acad. Sci. U.S.A.*, 98:14907–14912, 2001.

[189] D.C. White. Structure and function of myxobacteria cells and fruiting bodies. In E. Rosenberg, editor, *Myxobacteria: development and cell interactions*, pages 51–67. Springer-Verlag, 1984.

[190] D.C. White. Myxospore and fruiting body morphogenesis. In M. Dworkin and D. Kaiser, editors, *Myxobacteria II*. American Society for Microbiology, 1993.

[191] D.E. Whitworth, A.B. Holmes, A.G. Irvine, D.A. Hodgson, and D.J. Scanlan. Phosphate acquisition components of the *Myxococcus xanthus* Pho regulon are regulated by both phosphate availability and development. *J. Bacteriol.*, 190: 1997–2003, 2008.

[192] J. W. Wireman and M. Dworkin. Developmentally induced autolysis during fruiting body formation by *Myxococcus xanthus*. *J. Bacteriol.*, 129:796–802, 1977.

[193] S.S. Witkin and E. Rosenberg. Induction of morphogenesis by methionine starvation in *Myxococcus xanthus*: polyamine control. *J. Bacteriol.*, 103:641–649, 1970.

[194] S. Wolfram. *A new kind of science*, chapter 2: The crucial experiment. Wolfram Media, Inc, 2002.

[195] C. Wolgemuth. Force and flexibility of flailing myxobacteria. *Biophys. J.*, 89:945–950, 2005.

[196] C. Wolgemuth, E. Hoiczyk, D. Kaiser, and G. Oster. How myxobacteria glide. *Curr. Biol.*, 12:369–377, 2002.

[197] M. Wooldridge and N.R. Jennings. Pitfalls of agent-oriented development. In K.P. Sycara and M. Wooldridge, editors, *Proceedings of the second international conference on autonomous agents*, pages 385–391. Association for Computing Machinery, 1998.

[198] M. Wooldridge and N.R. Jennings. Software engineering with agents: pitfalls and pratfalls. *IEEE Internet Computing*, 3:20–27, 1999.

[199] F.Y. Wu. The Potts model. *Mod. Phys. Rev.*, 54:235–268, 1982.

[200] Y. Wu, Y. Jiang, D. Kaiser, and M.S. Alber. Social interactions in myxobacterial swarming. *PLoS Comput. Biol.*, 3:2546–2558, 2007.

[201] Y. Wu, N. Chen, M. Rissler, Y. Jiang, D. Kaiser, and M.S. Alber. CA models of myxobacterial swarming. In *Lecture Notes in Computer Science*, pages 192–203. Springer-Verlag, 2008.

[202] Y. Wu, D. Kaiser, and M.S. Alber. Periodic reversal of direction allows myxobacteria to swarm. *Proc. Natl. Acad. Sci. U.S.A.*, 106:1222–1227, 2009.

[203] H. Zhang, N.N. Rao, T. Shiba, and A. Kornberg. Inorganic polyphosphate in the social life of *Myxococcus xanthus*: Motility, development, and predation. *Proc. Natl. Acad. Sci. U.S.A.*, 102:13416–13420, 2005.

[204] D.R. Zusman. 'Frizzy' mutants: a new class of aggregation-defective developmental mutants of *Myxococcus xanthus*. *J. Bacteriol.*, 150:1430–1437, 1982.

[205] D.R. Zusman, A.E. Scott, Z. Yang, and J.R. Kirby. Chemosensory pathways, motility and development in *Myxococcus xanthus*. *Nat. Rev. Microbiol.*, 5:862–872, 2007.

Index

A

agents, 70

aggregation, *see* fruiting body

C

C++, 67, 70, 89

  pragma, 92

cell influx, 190

cellular automata, 32

  neighbourhoods, 35

cellular Potts model, 48

  cell sorting, 110

cheating, *see* myxobacteria

computation models, 27

concurrency, 89

critical region, 89

D

design patterns, 85

  decorator, 87

  factory method, 87

  interpreter, 89

  iterator, 88

  observer, 88

  proxy, 87

  singleton, 88

Dictyostelium discoideum, 18

F

FABCell, 66

  plug-ins, 99

  architecture, 73

  class diagrams, 73

  compiling, 251

  core libraries, 73

  implementation, 243

  interaction step, 79

  preconditions, 80

  representation of space, 76

  use cases, 69

  user interface, 96

  visualisation, 103

fibrils, 14

Fruit-Motilator, 188

  algorithm, 191

fruiting body, 18

fruiting body formation, 184
Frz transduction pathway, 122, 146
Frzilator, 233

G

game of life, 108
growth media, 52
  A1W, 52
  A1W-P, 52
  M1, 53
  M1-P, 53

H

Hamiltonian, 39, 156, 191
high performance computing, 70

I

implementation, 123

J

Java™, 70

L

lysis, 18

M

massively parallel processors, 90
MATLAB®, 233, 239
Monte Carlo, 143
Monte Carlo methods, 38
  equilibrium, 40
  Ising model, 41
  Metropolis algorithm, 45
Motilator, 114, 231, 239
motility, 13
multicellular, *see* fruiting body
myxobacteria, 8
  cancer treatment, 25
  cheating, 24
  inter-cellular signalling, 12
  life-cycle, 15
  measurement, 56
  resurrection, 55
  sporulation, 22, 198, 209
  stock solutions, 55
  stone restoration, 25

N

neighbourhoods, 81
  circular, 84
  Moore, 83
  Von Neumann, 83

O

off-lattice, 149

P

parallel computation, 89
parallel programming
  fork, 91
  MPI, 89
  OpenMP, 89, 91
  parbegin, 95
  pthreads, 89
phosphate, 50

phosphate sources, 52
Phys-Motilator, 146
pili, 14
potts model, 46

R

requirements analysis, 68
ripple formation, 171
rippling, 16

S

simulation volume, 189
slime mould, 18
software testing, 106
sporulation, *see* myxobacteria
stability analysis, 124
streaming, 17
symmetric multiprocessors, 89, 90
system energy
    adhesion, 194
    alignment, 158
    bending, 158
    climbing, 159, 195
    collision, 160
    gravitational, 161, 197
    propulsion, 161
    stretching, 163

U

UML, *see* unified modelling language
unified modelling language, 68
use cases, 68

V

vegetative cells, 15

W

Wolfram rule 110 cellular automaton, 243

Colophon

This thesis was written using TeXnicCenter 1.0RC [174]. The document was compiled using the MiKTeX 2.7 [144] distribution of LaTeX. The following LaTeX packages were used: `amsfonts`, `amsmath`, `avant`, `bm`, `colortbl`, `fancyhdr`, `fixltx2e`, `float`, `graphicx`, `natbib`, `mathpazo`, `rotating`, `setspace`, `titlesec` and `url`. Bibliographic information was processed by `BibTeX` and `makeindex` was used to typeset the index.

The thesis was formatted using a template created by the author. Document (excluding figures) titles, headings, captions and other meta-data were typeset in `Avant Garde` and the body text was typeset in `Palatino`.

Figures were created and edited using Adobe® Illustrator® CS3 and OpenOffice.org™ Draw 3.0.1 [123]. Graphical plots were created using MATLAB® 2006b. Certain three-dimensional figures were created using POV-Ray 3.6.1 [121]. Figure labels were typeset in `Arial`. Plots were typeset using `Helvetica` with additional meta data typeset in `Arial`. All figures were created as Portable Document Format (PDF) documents for inclusion in thesis and where possible are vector graphics.